(2)

DAVID W. TAYLOR NAVAL SHIP RESEARCH AND DEVELOPMENT CENTER

Bethesda, Maryland 20084

ZOG/VINSON TECHNOLOGY DEMONSTRATION PROJECT

SYSTEM DESCRIPTION

VOLUME II

by

Donald J. Schmelter
Ron Lupish

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

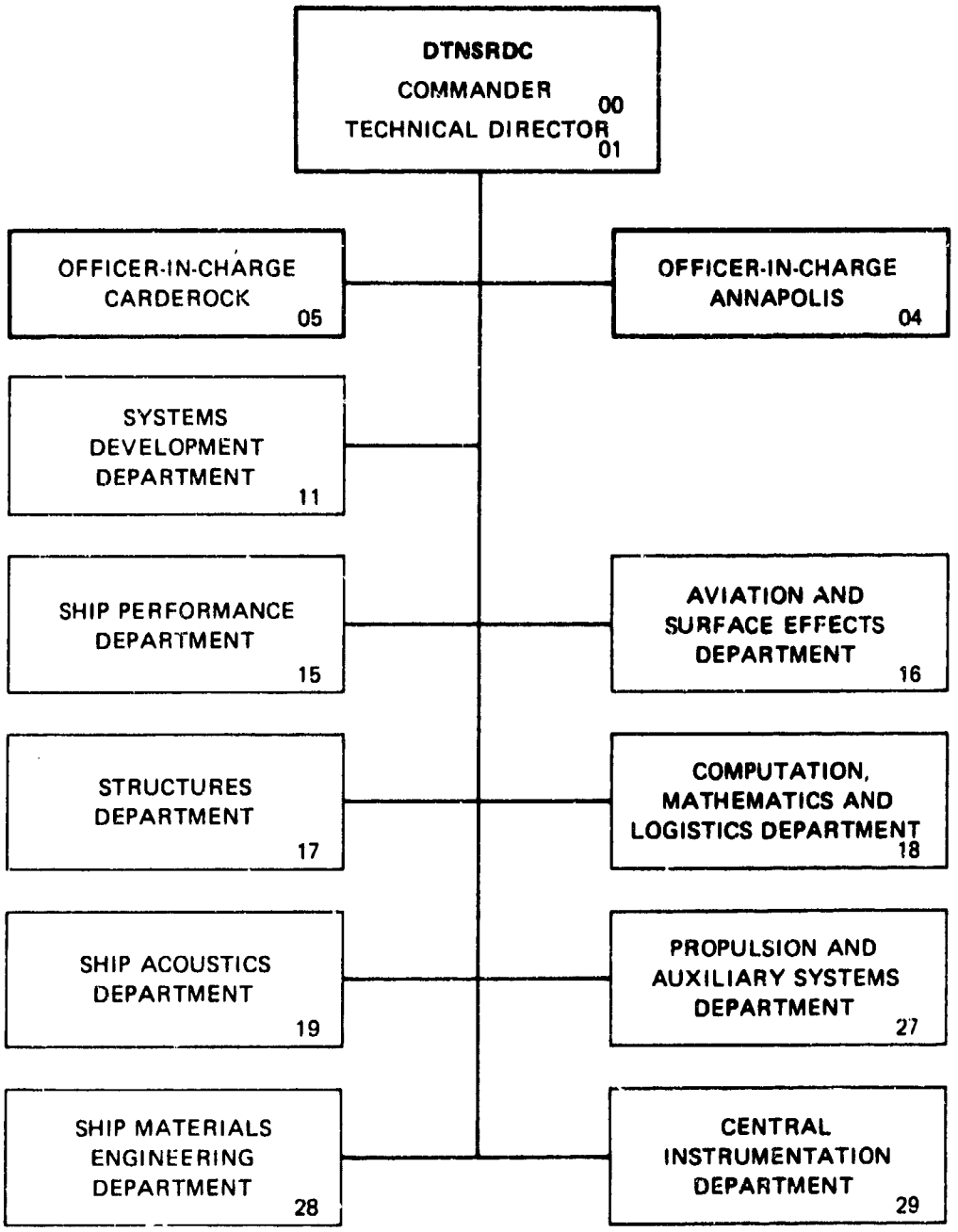
COMPUTATION, MATHEMATICS, AND LOGISTICS DEPARTMENT
DEPARTMENTAL REPORT

February 1985

DTNSRDC/CMLD-85/02

ZOG/VINSON TECHNOLOGY DEMONSTRATION PROJECT
SYSTEM DESCRIPTION, VOLUME II

MAJOR DTNSRDC ORGANIZATIONAL COMPONENTS



REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) DTNSRDC/CMLD-85/02	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION David Taylor Naval Ship R&D Center 6b OFFICE SYMBOL (*If applicable*) Code 1826	7a. NAME OF MONITORING ORGANIZATION
6c ADDRESS (*City, State, and ZIP Code*) Bethesda, MD 20084-5000	7b. ADDRESS (*City, State, and ZIP Code*)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Office of Naval Research 8b. OFFICE SYMBOL (*If applicable*) Code 270	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (*City, State, and ZIP Code*) Arlington, VA 22217	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 62763N; PROJECT NO RF63521; TASK NO.; WORK UNIT ACCESSION NO. 11826008

11 TITLE (*Include Security Classification*)
ZOG/VINSON TECHNOLOGY DEMONSTRATION PROJECT: SYSTEM DESCRIPTION, VOLUME [redacted] II

12 PERSONAL AUTHOR(S)
Donald J. Schmelter, Ron Lupish

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (*Year, Month, Day*)	15. PAGE COUNT
Final	FROM Mar 81 TO Oct 84	February 1985	300

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUB-GROUP	

19 ABSTRACT (*Continue on reverse if necessary and identify by block number*)

The ZOG system is a user-modifiable, menu-oriented, rapid response human-computer interface on a network of powerful workstations, the PERQ Systems Corporation's PERQ micro-computers. This document describes how the ZOG system operates. This is not a user's guide but a description of what is behind all of the menu creation and other standard features.

This document is divided into four separate and distinct volumes. It was written this way to best describe the total system while keeping volumes apart so as to make each one accessible without having to go through the others. The first volume is the ZOG System Description. The system description includes a description of:

An overview of the ZOG system, the initialization process, basic system flow, how accessing frames and subnets is accomplished, ZOG utilities, ZOG editors and ZOG agents.

The second volume is ZOG Structures. This volume lists all of the differenct structures used within the code that makes up ZOG.

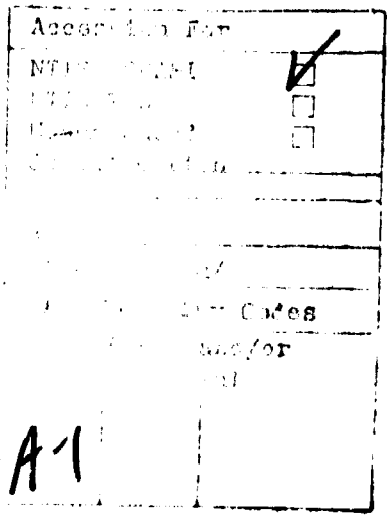
The third volume is ZOG Files. ZOG Files lists and describes all of the files that ZOG

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL Donald J. Schmelter	22b TELEPHONE (*Include Area Code*) (202) 227-1622 22c. OFFICE SYMBOL Code 1826

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

(Block 19) Continued
needs in order to run. These files are in addition to all of the source and executable files that make up the ZOG system.
The fourth volume is ZOG Modules. ZOG Modules goes through ZOG, module by module, describing what is going on.

Each of the four volumes has a different function. The system description is useful for an overall view of how ZOG functions. The structures volume is good for a quick reference of what all of the ZOG records and types are. The files volume is useful to see exactly what files ZOG needs and what they are used for. The modules volume is extremely useful for actually going into the pascal code and seeing how ZOG works on a module level.



ZOG Structures

Table of Contents

ZOG Structures

This subnet outlines the basic ZOG structures used throughout the ZOG system software.

- Structure of a ZOG Frame
- Backup Stack Structure
- Window Structure
- User Display text buffer
- Canvas Structure
- NetStack Record Structure
- ZOGNet Server Related Structures
- EtherNet Service Related Structures
- Editor Structures

1. Structure of a ZOG Frame

The information given here can be found in code form in ZOG Module NetDefs.

1.1. Pascal Record structure of ZOG frame : FTyp

The ZOG Frame record structure is defined in Module NetDefs as follows:

NextFr	FPTyp; Pointer to next frame
FrameID	FIdTyp; Frame Id
Owners	UsrIdPTyp; List of Frame owners
CrDate	long; Creation Date of frame
Modifier	UsrIdTyp; Last modifier of frame
ModDate	long; Date of last modification
ModTime	long; Time of last modification
Version	integer; Frame Version number
Prot	ProtTyp; Protection code of Frame
AgCrBit	boolean; True if created by agent
AgModBit	boolean; True if modified by agent
Title	SelPTyp; Pointer to frame title
Text	SelPTyp; Pointer to frame text

Options	SelPTyp; Pointer to frame options
LPads	SelPTyp; Pointer to frame local pads
GPad	FldTyp; Frame ID of global pads frame
Comment	FsPTyp; Pointer to comment strings
Accessor	Fs15PTyp; Pointer to Accessor Frame Ids -- not used

1.2. ZOG TypeDefs

Many of these structures used in defining a ZOG Frame are also used throughout the ZOG system. Some of these structures are accessed via pointers. These types are discussed below:

- FPTyp = ↑FTyp; Pointer to a Frame Record
- FldTyp = string[15]; Stores Frame Ids
- UsrIdTyp = string[15]; Stores user Names
- ProtTyp = integer; UnProt, ModProt, WrProt, RdProt
- PosTyp = integer; For storing row and column information
- SelPTyp Pointer to a selection record
- FsPTyp Pointer to a linked list of strings
- Fs15PTyp and UsrIdPTyp Pointers to linked lists of short strings

1.2.1. Frame Pointer

The definition of a frame pointer is simply as a pointer to a frame record. Examples of frame pointer variables used throughout ZOG are FPX, EdFP, and SledFP.

1.2.1.1 FPTyp = ↑FTyp; where FTyp is the frame record definition

1.2.1.2 FTyp = Record

See Section 1.1, page p. 1.

1.2.2. Frame ID

The Frame ID is defined within ZOG Pascal code to be a string of length 15. It is composed of the concatenation of a Subnet Name with an integer in character form. Since the maximum length of a FrameId is 15, this implies a maximum length (in characters) of a Subnet Name of 11 characters, which will provide for a subnet of up to 9999 frames. Although this is the case, the SubnetID type is also defined as a string of length 15. The formal definitions are:

FidTyp	string[15]; Frame Id Type

SidTyp string[15]; Subnet Id (name) Type

1.2.3. User Ids

User IDs are variables used to store the name of the currently logged on user. The definition is a string of length 15, which imposes a limitation on the length of the user names which can be added to the PERQ user list via UserControl. User Ids are used to identify the creator and modifier of frames.

1.2.4. Frame Protection

Every frame is assigned a protection, the default being no protection at all. The protection variable within a frame record is defined as an integer. Frames can be protected against modification, writing and reading. Frames can only be protected by the owner against the "rest of the world"; there is no notion of group access/protection privileges other than multiple-owners.

Frame protection is coded as:

0 No Protection; UnProt = 0

1 Protection from being modified; ModProt = 1

2 Protection from being written; WrProt = 2

3 Protection from being read; RdProt = 3

1.2.5. PosTyp = integer; For storing row and column information

1.2.6. SelPTyp

SelPTyp is a pointer to a linked list of item records within a frame. Its name implies a "selection" record, although these structures are used for frame text and title as well. Items marked below with a "*" are not applicable for frame text and title.

K char; * Selection character

NF FldTyp; * Next frame

Text FsPTyp; Pointer to linked list of item's text

Row PosTyp; Item Row Position

Column PosTyp; Item's Column Position

L0 PosTyp; Item's Minimum Row Position - "top edge"

C0 PosTyp; Item's Minimum Column Position - "left edge"

L1 PosTyp; Item's Maximum Row Position - "bottom edge"

C1 PosTyp; Item's Maximum Column Position - "right edge"

Action FsPTyp; Item's associated Action(s) (in text form)

Expand FsPTyp; Item's associated Expansion Area, for misc. use

ExtraFlds FsPTyp; Fields not currently used - possible later use

PrevSel SelPTyp; * Pointer to previous item

NextSel SelPTyp; * Pointer to next item

1.2.7. FsPTyp

FsPTyp is a pointer to a doubly linked list of "frame" strings, although it is used throughout ZOG as a pointer to a linked list of generalized strings of (default) length 80. Its Pascal definition is FsPTyp = ↑FsTyp, where FsTyp is a record defined as:

text string; string is default length of 80 characters

prev FsPTyp;

next FsPTyp;

1.2.8. Fs15PTyp and UsrIdPTyp

Fs15PTyp and UsrIdPTyp are pointers to short strings (i.e., strings of length 15, which are used within ZOG for a variety of purposes). They are used throughout ZOG as general pointers to a linked list of short strings. They are both defined as = ↑Fs15Typ, where Fs15Typ is a record defined as:

text string[15]

prev Fs15PTyp

next Fs15PTyp

2. Backup Stack Structure

ZOG maintains an ongoing list of frames visited on a stack structure while the user is traversing ZOG nets. This stack is basically a doubly linked list structure, for which added storage is dynamically allocated as necessary. Entries are removed (popped) from this structure as the user goes "back" up the tree of frames.

This structure is defined in module ZWind, but is maintained by module ZBack. Its definition is:

2.1. Pascal Definition of the ZOG Backup Stack

BackPTyp = ↑BackTyp

BackTyp = RECORD

FrameId FIdTyp; Frame ID of frame on the stack

SelCh char; Selection character that was used in departing from this frame

Nxt BackPTyp; Pointer to next record on stack

Prv BackPTyp; Pointer to previous record on stack

3. Window Structure

The ZOG Window Management Module, ZWind, keeps track of all the information necessary to maintain the displays of frames in each of the ZOG windows. It does this by maintaining a record for each window. These dynamically allocated records contain such information as current frame id and version number, a pointer to current frame record, global pads frame id and pointer to GPads record. It also keeps track of the top (i.e., the last or most recent) frames on the Backup and Mark stacks.

This record keeps track of which alternate global pads frame (if any) is being displayed.

3.1. Pascal Definition of the ZOG Window Structure

WindPTyp = ↑WindTyp; Pointer to the window structure

WindTyp = RECORD

Number	integer; This window's ZOG Window number(1 or 2)
Canv	integer; ZOG Canvas number for this window
BackP	BackPTyp; Pointer to top frame on Backup Stack
MarkP	BackPTyp; Pointer to top frame on Mark Stack
FrameId	FldTyp; Frame ID of current frame
Version	integer; Version number of current frame
SelChar	char; Char selected to get to current frame
GFrameId	FldTyp; Frame ID of current Global Pad Frame
DisFrameId	FldTyp; Frame ID of frame currently displayed
FPX	FPTyp; Pointer to current frame record
GFPX	FPTyp; Pointer to current GPad frame record
SecSig	boolean; True if secondary copy read for display
AltGPads	boolean; True if alternate global pads in use
AltGpadFid	FldTyp; FrameId of alternate global pad frame
MarkCnt	integer; Number of frames "marked"

4. User Display Text Buffer:

The (full-screen) User Display Text Buffer is a dynamically allocated array of strings. At the current time, the number of elements (i.e., lines of text) in this array is 77, as defined by the local constant, FullSz. This array is pointed to by local pointer variable, UsrDspP, which is created with a call to "new" in InitUser. All accesses to text in either the short or full-length user display are referenced via this variable.

PASCAL definitions are given below:

- UsrDspP : UsrDspPTyp;
- UsrDspPTyp = ↑UsrDspTyp;
- UsrDspTyp = Array[0..FullSz] of string;

5. Canvas Structures

These records are used to keep track of window types and "events" within a window. These structures are defined in ZCanvas.Defs.

5.1. Canvas Type

This is used to monitor which canvas is the current one.

ZOG1Canvas	1; Canvas for the top frame display window
ZOG2Canvas	2; Canvas for the bottom frame display window
UserCanvas	3; Small user display window, at bottom of screen
FullCanvas	4; Full screen user display window
DoubleCanvas	5; Double-sized canvas for displaying big frames
CanvasTyp	ZOG1Canvas..DoubleCanvas;

5.2. Canvas Event Record

CanvEvPtr = ↑CanvEvtyp

CanvEvTyp = record

Ch	char; Input character, from keyboard, or mouse char
x,y	integer; X,Y Mouse coordinates in pixels (768 x 1024)
Row,Col	integer; Mouse coordinates in chars, relative to window
KeySig	boolean; True if keyboard or mouse input, false otherwise (i.e., if only mouse movement)
Canv	CanvasTyp; Canvas from which the event arose
ChgSig	boolean; True if a change of windows occurred
NextEv	CanvEvPtr; For queueing up canvas events

6. NetStack Record Structure

FStkTyp = record

This structure was used extensively throughout ZOG for keeping track of frames traversed, particularly in agents which need to traverse trees exhaustively. However, since it doesn't stack frame records, popping the stack requires re-reading previously read frames. Since this requires disk I/O, it proved to be much faster using the stack declared in StackLib.

NetStack exports record variable FStkX as a global frame stack for use in ZOG with pointer variable FPX for the pointer to the current frame. Routines in NetStack read frames into this record, and use FPX as the default frame record pointer.

FrameId	FIdTyp; Frame ID of top frame on stack
FIdP	Fs15PTyp; Pointer to list of saved frame Ids
FP	FPTyp; Pointer to frame record of top frame on stack

7. ZOGNet Server Related Structures

7.1. Hashed table of subnets

This table is EXPORTed from Module ZOGNetServer. It is used to see if a subnet requested by a user already exists and to hold entries for new subnets created.

Subnets : Array[0..MaxHashIndex-1] of pSubnetTyp

pSubnetTyp = ↑SubnetTyp

SubnetTyp = record

NextSubnet	pSubnetTyp; Pointer to next hashed subnet name
SubnetID	SIDTyp;
MatchID	SIDTyp; SubnetID converted to all Upper Case
PrimeNode	NodeTyp; NodeTyp is an integer
SecCnt	integer; Count of number of secondary nodes
SecNodes	SNodesTyp; Array[1..MaxSecondary] of NodeTyp where MaxSecondary = 5 (NetDefs)
Opened	boolean;

7.2. Hashed Table of subnets (Local Subnet Index).

SnTable represents the local subnet index which is private to module NetServ. This second subnet index contains low level information necessary in accessing subnet files.

SnTable : Array [0..SnHashMx-1] of SnRecPTyp

SnRecPTyp = ↑SnRecTyp

SnRecTyp = record

UpSid	SidTyp; Subnet ID converted to upper case
Sid	SidTyp; Subnet ID with true case(for CrF,CrFr)
PrimeSig	Boolean; If true the record is for a primary subnet.
Nxt;	SnRecPTyp; Ptr to the next subnet record
FilID	FileID; file system file ID
SzPg	Long; size of the file in pages

7.3. List of open frames

The list of open frames is private to module NetServ, thus each machine has its own list of open frames which will consist of the frames opened on that particular machine. These frames can be opened locally and from a remote machine.

OpnTopP : OpnRecPTyp Ptr to open frame record list

OpnRecPTyp = ↑OpnRecTyp

OpnRecTyp = record

UserPort	EtherAddress; Identifies user that opened frame
UpSid	SidTyp; Subnet ID of opened frame(string15)
Fnr	integer; Relative frame number of opened frame
SnRecP	SnRecPTyp Ptr to subnet record, used by ClsF –
BodySzPg	integer; size of frame body in pages
Nxt	OpnRecPTyp; Ptr to next open frame

8. EtherNet Service Related Structures

8.1. List of available EtherNet Servers

This structure holds the list of PERQs on the current ZOG distributed network. There is room on this list for MaxZOGServer (currently 256) EtherNet Servers (i.e. remote PERQs). This structure is EXPORTed from Module ZOGNetServer.

Servers : array[0..MaxZOGServer-1] of pServerStatus

pServerStatus = ↑ServerStatus

ServerStatus = record

ServerAddr EtherAddress;

ServerUp boolean;

ServerName string15; Name of machine from Net.Servers

8.2. ZOG Ethernet message and buffer types

This type is used in ethernet communication when the machines are sending or receiving records to one another, and when the machines are sending buffers of information to one another. The actual request and reply packets are recast as ZOG ethernet message types to be sent over the ethernet. These declarations can be found in Module ZogMsg.

These types were created to allow requests and replys to be sent over the ethernet using only one type of record. Recasting a request or reply packet to a message or buffer type means that only one routine is necessary for sending and receiving the various record types used for ethernet communication, even though different processes (reading, opening, closing, etc.) require different information. The message and buffer types can be viewed as a black box of information where the sending and receiving routines know the structure of the information contained, but the actual packet carrying that information does not know of its structure.

8.2.1. ZOGMsgTyp

ZOGMsgTyp's are used in sending and receiving records.

ZOGMsgPTyp = ↑ZOGMsgTyp

ZOGMsgTyp = packed record

Id Integer; Message Identifier

LocalAddr EthernetAddress;

RemoteAddr EthernetAddress;

RemoteName String15; Name of remote Ethernet machine. Used only when receiving messages.

GR GeneralReturn; Used only i reply messages to return a possible error message

Body packed array [0..ZogMsgBodySiz-1] of integer;

8.2.2. ZogBufTyp

When buffers are sent or received over the ethernet they are recast as BufPTyp. This is a generic pointer (a PERQ Pascal extension), which can be used to point at anything (there are restrictions on its use, refer to the section on Pascal extensions in the PERQ Software reference manual), and is used here to point to a ZOG page buffer.

```
BufPTyp = pointer; Ptr to Zog page buffer
ZOGPagePTyp = ↑ZogPageTyp
ZOGPageTyp = packed array [1..PageWordSize 256] of integer
ZOGBufPTyp = ↑ZOGBufTyp
ZOGBufTyp = packed record Zog Message Buffer type
```

LocalAddr	EthernetAddress
RemoteAddr	EthernetAddress
RemoteName	String15; name of remote ethernet machine
Size	integer; 1 indicates a single page, 2 = 2 pages
PgNum	integer; counts pages for multiple pg. transfer
Page1	ZOGPageTyp
Page2	ZOGPageTyp

8.3. Ethernet Request packet records

Module ZogMsgDefs contains the declarations of all of the Ethernet request and reply packets. For the most part the types contained in this module are the same with small variations in the records, due to the function of a particular type. Only two of the types will be shown here. The information contained in these records allows the sending and receiving routines to accomplish their purpose (i.e. opening, closing, etc.). This information is loaded into the appropriate record, recast to a message or buffer type and sent over the ethernet. On the remote machine, the appropriate receiving routine knows the structure of the information in the request or reply packet, so it knows where to get the information it needs.

8.3.1. Open Frame Request Packet.

```
OpnF0PTyp = ↑OpnF0Typ
OpnF0Typ = packed record
```

Id	integer; Constant identifier in ZogMsgDefs
LocalAddr	EthernetAddress;
RemoteAddr	EtherNetAddress;

RemoteName	String15;
GR	GeneralReturn; used in return packet
Name	UsrIdTyp;
AgentFlag	Boolean;
Sid	SidTyp; Subnet ID
PrimeNode	NodeTyp;
SecCnt	integer;
SecNodes	SNodesTyp;
FrNum	integer;

8.3.2. Open Frame Reply Packet

OpnF1PTyp = ↑OpnF1Typ

OpnF1Typ = packed record

Id	integer; Constant identfier from ZogMsgDefs
LocalAddr	EthernetAddress;
RemoteAddr	EthernetAddress;
RemoteName	String15;
GR	GeneralReturn; Return code from remote node
FHBCnt	long; Count of Frame Header pages
FBCnt	long; Count of Frame Body pages

8.3.3. Close Frame Request Packet

ClsF0PTyp = ↑ClsF0Typ

ClsF0Typ = packed record

Id	integer; Constant identifier in ZogMsgDefs
LocalAddr	EthernetAddress;
RemoteAddr	EtherNetAddress;
RemoteName	String15;
GR	GeneralReturn; used in return packet
Name	UsrIdTyp;
AgentFlag	Boolean;
Sid	SidTyp; Subnet ID
FrNum	integer;

FBCnt long; Count of Frame Body Pages

8.3.4. Close Frame Reply Packet 1

ClsF1PTyp = ↑ClsF1Typ

ClsF1Typ = packed record

Id integer; Constant identfier from ZogMsgDefs

LocalAddr EtherAddress;

RemoteAddr EtherAddress;

RemoteName String15;

GR GeneralReturn; Return code from remote node

8.3.5. Close Frame Reply Packet 2

ClsF2PTyp = ↑ClsF2Typ

ClsF2Typ = packed record

Id integer; Constant identfier from ZogMsgDefs

LocalAddr EthernetAddress;

RemoteAddr EthernetAddress;

RemoteName String15;

GR GeneralReturn; Return code from remote node

FHBCnt long; Count of Frame Header pages

9. Editor Structures

These structures are global within ZED, the ZOG frame Editor.

9.1. Delete Buffer

This buffer stores characters as they are deleted from text anywhere in the frame. It is used for moving and copying text within and between frames. The definition and exported (global) variables are in Module ZEDDefs.

BufP :BufStrP; Global (ZED-wide) delete buffer pointer

BufStrP = ↑BufStr

BufStr = RECORD

Length integer;

Content array[1..1760] of char;

9.2. Current text position

This record is used to maintain current position information about the current item in a frame being edited. The record type is defined in Module ZDspInc; the Global variable, CItem, however, is exported from Module ZEDDefs.

CItem : TxtPosTyp; Global (ZED-wide) Current Item record

TxtPosTyp = record

TxtP	SelPTyp; Pointer to current item
Row	integer; Current relative row position within item
Column	integer; Current relative column position within item
CurStrP	FsPTyp; Pointer to current string within item text

9.3. Item types

This is an enumerated variable to keep track of the type of the current item. It is defined and exported from Module ZEDDefs.

ItemType	(ITitle, IText, IOptions, IPads);
TypeItem	ItemType;

9.4. Types for maintaining selected text — *Not Currently Used*

These structures were defined for the proposed enhancement of ZED, to include "Pepper-like" selection of text for deletion or moving around. This text could be in the current frame or any other frame.

9.4.1. StrInfo = Record A structure to store selected substrings

Row, LCol, RCol	integer; To store selected string starting and ending points
Str	string ; NOTE: can only store up to 80 chars

9.4.2. Selections = Record The structure which holds the entire selected string, and related information

Fid	FIdTyp; The frame which contains the selected str
SelP	SelPTyp; Points to the selection containing the selected string
Version, Row, Col	integer; Holds the frame version number, and row/column information
Cnts	integer;
Lines	Array[1..10] of StrInfo; The text itself

9.5. Back Room Editor Type for options

This record allows the "redefinition" of a single line/option in an AirPlan input frame into a number (up to MaxOpts, currently 10) individual options while the frame is being edited. The option reverts back to a single option before being written out. Note: Each option on an AirPlan input frame is identical in layout; hence only the record structure below is necessary, since it applies to all rows (options) in the input frame.

Field : FieldTyp

FieldTyp = record

Column	Array[1..MaxOpts] of integer; Stores starting position of each redefined option in a given row.
Len	Array[1..MaxOpts] of integer; Stores the length of each corresponding option.

ZOG Files

Table of Contents

ZOG Files

In order for ZOG to run, certain files (other than subnets) must be available.

1. Input Files for ZOG

The following files provide information necessary for ZOG to run. These files are opened and read as part of ZOG's initialization process. All of the information read is stored in memory for fast access as ZOG executes normally.

- :zognet>subnet.index
- :boot>Ethernet.Names
- :zognet>net.servers
- :zognet>zog.animate
- :zognet>sec.default
- :zognet>zognet.setup
- :zognet>help.frame

1.1. :zognet>subnet.index

Subnet.Index is a listing of all the subnets available to ZOG. Each PERQ in the network must have its own local copy of subnet.index minimally listing the *core subnets*. At any point in time, only the *master* PERQ's subnet.index file may be a complete list of all subnets. It will be necessary to periodically update each subnet.index file on all other PERQs in the network with the current copy of subnet.index from the *master* PERQ. This is the responsibility of the local ZOG system maintainer.

During initialization of ZOG, subnet.index is read. Each subnet name is hashed into an internal table maintained and accessed in Module ZOGNetServer. When a frame from some subnet is requested, procedures in ZOGNetServer will do a lookup in the local hashed table of subnet.index to find out if the subnet exists, and if so, on which PERQ it resides. If the subnet name is not found in the local table, a request will be sent to the *master* for this information.

1.1.1. Structure of subnet.index

This file is a seqential list of subnet names, with each subnet name followed by information specifying the nodes (i.e., remote PERQs) on which the subnet resides. Individual PERQs are identified by a node number (its sequence in file :zognet>Net.Servers). Entries in Subnet.Index look like:

```
FOO
2 0
BAZ
1 1 6
```

where FOO and BAZ are legitimate subnet names. The numbers below each subnet name indicate the following:

- First number is the Node number of the PERQ on which the primary copy of the subnet resides
- Second number indicates the number of secondary copies of the subnet (up to a maximum of MaxSecondary = 5)
- Additional number(s) indicate the PERQ(s) on which the secondary copies reside

1.2. :boot>Ethernet.Names

File Ethernet.Names contains the Ethernet name of the local PERQ. Users can edit this file to change the name of their machine; however, this should not be done unless the :zognet>Net.Servers files on ALL other PERQs in the network are changed correspondingly. If this rule is not followed, communications with other PERQs on the network will not be possible.

Only the first line of this file is of importance; any other lines in it are ignored by ZOG. Thus, other names for the local machine could be stored here for later reference.

1.3. :zognet>net.servers

The Net.Servers file is a list of all PERQs on the current ZOG network, by name. This file is read sequentially in ZOGNetServers.BuildServers to build the global ServersStatus table. The table is accessed with integers from 0 to (N - 1) where N is the number of PERQs listed. The *master* PERQ is always the listed first, and thus is number 0. The node numbers associated with each PERQ listed in this file are also the node numbers used in file :zognet>subnet.index.

The name of each PERQ is contained in file :boot>EtherNet.Names. If there is more than 1 line in this file, the name of the PERQ is the name (ASCII string) contained in line 1 of this file.

It is very important that all PERQs in the ZOG network have identical copies of this file, otherwise, serious inconsistencies in machines' subnet.index files will develop.

1.4. :zognet>zog.animate

ZOG.animate is used to contain the default mouse cursors of the solid arrow, and the cursor used to indicate that an Ethernet event is occurring (i.e., that the user is temporarily locked out of ZOG for just a moment). This second cursor is usually the hollow arrow.

Users can use the special PERQ utility, CursDesign, to edit this file, so as to create a new set of default cursors for ZOG to use.

If this file is not present, ZOG will only use the POS default mouse cursor of the solid arrow. No indication is given that the file can't be found, but Ethernet interrupts will not be noticed by the user other than the keyboard being temporarily locked.

1.5. :zognet>sec.default

This file specifies the default number of the PERQ(s) to be used by this machine for secondary copies of its subnets. Its format is similar to that used in :zognet>subnet.index: The first number specifies the number of secondary copies for each of the local PERQ's subnets there are to be, up to a maximum of five. The remaining number(s) specify the PERQ(s) these secondary (or backup!) subnets are to reside on. The PERQ node numbers are specified in file :zognet>Net.Servers.

Secondary subnet files are identified by having the file name <Subnet>.sec. These may be located in partition :zognet on some machines, and in partition :second on others (see :zognet>zognet.setup for details).

Examples of possible contents of sec.default are:

0	no secondary copies to be made at all
1 7	1 secondary copy to be maintained on PERQ number 7
3 1 3 5	3 secondary copies of each subnet to be maintained, one copy on PERQ 1, one on PERQ 3, and one on PERQ 5.

1.6. :zognet>zognet.setup

This file contains a boolean (i.e., the string *True* or *False*) as its only entry.

A value of true implies that all subnets, including local and secondary copies of subnets, are located in partition :zognet.

A value of false implies that local subnets will be in partition :local, and secondary subnets will be stored in partition :second. All others will still reside in partition :zognet or in partition :primary, if it

exists.

1.7. :zognet>help.frame

Help.Frame specifies the frame to be displayed in the *other* window whenever the "help" global pad is selected. It's default is Help1.

2. ZOG's Output Files

ZOG's output files are used to record information on disk as an aid in trouble-shooting or debugging the system.

- :zog>log>zog.log<n>
- :zog>log>exception.log

2.1. :zog>log>zog.log<n>

This set of files (i.e., zog.log, zog.log1, zog.log2, zog.log3) keep a four-deep log of all the user display messages displayed during a ZOG run. :zog>log>zog.log is the current log file, the remaining zog.log<n> files are the logs of the last, last but one, and last but two ZOG runs, respectively. In addition, login/initialization time is recorded, as is logout/ending time. Also, if ZOG aborts abnormally, the Pascal stack dump information is also recorded in zog.log.

These files are created and maintained by procedures in Module ZLogFile.

2.2. :zog>log>exception.log

Occasionally ZOG may abort due to an uncaught Pascal-generated exception, such as a *string too long* exception. Normal procedure on the PERQ is for the running program to abort, and a *stack dump* of the Pascal procedures which were invoked when the exception was generated to be displayed on the screen. In ZOG, however, there is an All exception handler to catch these exceptions. The purpose of this handler is twofold: to record the exception-generated stack dump in the file :zog>log>exception.log (and in :zog>log>zog.log); and to exit ZOG "gracefully", i.e., in a controlled fashion.

This file is created, or appended to, via Procedure ZOGDump in Module ZDump.

ZOG Modules

Table of Contents

The ZOG Code Modules are listed below by functional category.

The following options name modules and point to frames which list the exported procedures/functions contained in those modules.

- ZOG System Modules
- ZOG NetServer Modules
- ZOG Editor Modules
- ZOG Agents Modules
- ZOG AirPlan Modules
- PERQ Operating System Modules IMPORTed by ZOG

1. ZOG System Modules

1.1. Basic System

1.1.1. Module ZBack

This module maintains the ZOG system backup list of frames visited, which is used in implementing the back, next, prev and ret global pads.

InitBack	Initialize Backup List Structure (and ZMark Module)
SavBack	Push currently displayed frame and selection on backup stack (ZBack.InsBack)
GoBack	Pop top frame on backup stack (ZBack.DelBack, ErBack)
XPop1Back	Pop up one level on the backup stack (ZBack.DelBack, ErBack)
XPopBack	Pop backup stack to given frame id
XClrBack	Clear entire backup list
XGoBack	Pop top frame on backup list, and display it - "back"
XRetBack	Display pseudo-frame listing frames on backup list - "ret"
XNext	Pop top frame on backup list, and display frame id pointed to by next option, if it exists - "next"

1.1.2. Program ZOG

The Main ZOG Program. Consists of a Main routine which calls Procedure ZOGMain, the "real" main ZOG Procedure. This module also contains the upper-level exception handlers and the Exiting and Logging-in invoking procedures.

SuspZOG	Suspend ZOG execution, saving current state

ResZOG Resume normal ZOG execution from saved state

ReLogZOG Determine if user "really" wants to log off from ZOG

ReInitZOG Determine if user "really" wants to exit ZOG. Calls ZInitExit.ExitZOG to perform controlled shutdown.

ZOGMain Main ZOG loop. Calls initialization routine, then loops on character input, character processing sequence.

1.1.3. Module ZOGVersion

This module supplies the current ZOG version number to the rest of ZOG.

1.1.4. Module ZParse

Contains some elementary parsing routines for the original ZOG command line invocation from the shell. Also used within ZOG for command line parsing.

InitParse If ZOG is declared (in user's profile file) to be the current shell, then get any switches passed to ZOG via GetArg, below

ProSwitches Process user-input switches from command line

GetArg Recursively obtains switches from command line; if specified argument is missing, will prompt for missing argument with caller supplied string

GetOptArg Recursively obtains switches from command line; if specified argument is missing, use passed default value

GetRemArg Get remaining arguements from the user command line

1.1.5. Module ZSel

Module ZSel contains 5 exported routines necessary to do selection processing. These are:

Procedure GetSel given a selection character, this routine returns a pointer to the corresponding selection, if it exists

Procedure EvalSel given a pointer to a selection on a frame, this routine will either go to the frame linked to the selection (and execute that frame's action), do the selection action, if there is one, or initiate top-down frame creation from the selection.

Procedure OutS given a character input by the user, this routine will interpret it as an action string or as a selection to be processed.

Procedure ReturnSel returns the selection character or control character input by the user.

Function GTchSel returns pointer to selection selected by mouse.

1.2. Initialization and Exiting

1.2.1. Module ZInitExit:

InitZOG Initialize all the variables and pointers in all of the modules that make up ZOG.

ExitZOG Clean up variables in preparation to exit ZOG.

LogOffZOG Log off one user and log in another.

1.2.2. ZInitOthers

This Module has one procedure which simply calls the remaining initialization procedures, in order, for the rest of the ZOG sytem modules. This procedure could not be a part of ZInitExit due to the PERQ Pascal compiler restriction on the number of imports allowed. Hence, ZInitExit InitZOG had to call ZInitOthers InitOthers, where ZInitOthers imports the rest of the system modules needed for initialization.

Procedure InitOthers
exported

1.2.3. ZLogin

ZLogin is the login program. It is called at both boot time and anytime a Login command is executed.

Procedure DoZOGLogIn
exported

1.3. System Level Libraries

- BaseLib
- FsString
- NetDefs
- NetInsert
- NetLib
- NetMakeDel
- NetOption
- NetPERQCodes
- NetStack
- NetString

1.3.1. BaseLib

BaseLib is a collection of routines that are needed by the basic ZOG system and the ZOG Net Server process. It's purpose is to avoid duplicating code for both processes.

1.3.1.1 Initialization routines

Procedure IniBaseLib
Initialize internal variables and pointers

Procedure IniFHP Initialize the frame header record structure

1.3.1.2 Test Functions

Function TIsUc Test to see if a character is uppe case

Function TIsLc Test to see if a character is lower case

Function TIsAlph Test to see if a character is Alphanumeric

Function TIsDi Test to see if a character is a digit

Function TOwnF Test to see if a given user Id is one of the owners of a frame

Function TSidValid Test to see if a subnet Id is valid

Function TProtValid
Test to see if given protection is valid

Function TUsrIdValid
Test to see if a given user Id is valid

1.3.1.3 Convert routines

Procedure CvIntStr
Convert an integer to a string

Function CvStrInt Convert a string to an integer

Procedure CvLongStr
Convert a long integer to a string

Function CvStrLong
Convert a string to a long integer

FunctionCvMonStrInt
Convert a string to a valid month integer

Function CvDatStrInt
Convert a date string into internal integer format

Function CvTimStrInt
Convert a time in string format into a long integer

1.3.1.4 Get Functions

Procedure GTimStr

Get the time of day in HH:MM:SS format

Function GTimInt Get time in milliseconds since midnight

Procedure GDatStr

Get today's date in DD MM YY format

Function GDatInt Get today's date in internal integer format

Function GEqFs15P

Get an entry on a frame string15 linked list matching a given string

1.3.1.5 Make-Delete procedures

Procedure SavFs15P

Put a frame string15 record structure on the save list

Procedure SavFHP Put a frame Header record structure on the save list

Procedure RelFs15P

Release a frame string15 record from use

Procedure RelFH Release contents of a frame header record structure

Procedure ClrFHP Clear contents of a frame header record structure

Function CrFs15P Create a frame string15 record structure

Function CrFHP Create a frame header record structure

Procedure DelFs15l

Delete a frame string15 record structure from a frame string15 record linked list

Procedure InsbFs15l

Insert a frame string15 record a the beginning of a frame string15 linked list

Procedure InseFs15l

Insert a frame string15 record at the end of a frame string15 linked list

1.3.1.6 Miscellaneous procedures

Procedure AppStrFile

Append a string to the end of a file

Function ParseLine

Get a line of info from an internal frame buffer that contains a frame in external bh frame storage format

Procedure ParseFH

Transforms the external bh form of a frame header into the internal frame record structure during the frame read process

1.3.2. FsString

Module FsString impliments the frame string manipulation routines for the ZOG system.Frame strings are record structures with three elements. The first element is a string[80] followed by two pointers. The pointers allow the frame strings to be put on doubly linked lists. All of the PERQ Pascal string manipulation routines in PERQ_String have been duplicated for the frame strings in this module.

1.3.2.1 Length and Write routines

Function Fs – Length
Get the length of a frame string

Function Fs – Lines
Count the number of lines in a frame string

Procedure WrFsFile
Write a frame string to a file

Procedure WrFsXFile
Write a frame string to a file beginning with the ith character; where i is given in the call

Procedure WrFs Write a frame string to the standard output

1.3.2.2 Convert routines

Procedure CvFsStr Convert a frame string to a Pascal character string

Function CvStrFs Convert a character string into a frame string

Procedure Fs – ConvUpper
Convert a frame string to all upper case letters

Procedure Fs – ConvLower
Convert a frame string to all lower case letters

1.3.2.3 Basic String routines

Procedure Fs – Adjust
Change the dynamic length of a frame string

Function Fs – Concat
Concatenate two frame strings together

Function Fs – SubStr
Return a subportion of a frame string as a character string

Procedure Fs – Delete
Remove characters from a frame string

Procedure Fs – Insert
Insert a string into a frame string

1.3.2.4 Position and Append routines

Function Fs – Pos Find the position of a pattern in a given frame string

Function Fs – PosC
find the postion of a char in a given frame string

Function Fs – RevPosC
Find the position of the last occurance of a pattern in a given frame string

Procedure Fs – AppendString
Append one frame string to the end of another frmae string

Procedure Fs – CAppend
Append a character to the end of a frame string

1.3.3. NetDefs

Module NetDefs contains all of the basic ZOG system definitions. These include all of the signal constants, basic type declarations, control character constants, declaration of protection types and various other constant declarations that are used throughout ZOG. NetDefs does not export any procedures.

1.3.4. NetInsert

Module NetInsert impliments all the list handling routines for frame string pointer types(Fsl), frame string15 pointer types (Fs15i) and selection list pointer types (Sell). Each routine inserts a record into a list of records either at the beginning (Insb), prior to the record already on the list (Insp), after a record already on the list (Insa) or at the end of the list (Inse).

1.3.4.1 Insert at the beinning of a list utilities

Procedure InsbFsl Insert a frame string record at the beginning of a f frame string record linked list

Procedure Insbell Insert a selection record at the beginning of a selection record linked list

1.3.4.2 Insert prior to an object on a list utilities

Procedure InspFs15l
Insert a frame string15 record prior to a given frame string record that is on a linked list of frame string15 records

Procedure InspFsl Insert a frame string record prior to a given frame string record that is on a linked list of frame string records

Procedure InspSell
Insert a selection record prior to a given selection recor that is on a linked list of selection records

1.3.4.3 Insert after an object on a list utilities

Procedure InsaFs15I
Insert a frame string15 record after a given frame string15 record that is on a linked list of frame string15 records

Procedure InsaFsI Insert a frame string record after a given frame string record that is on a linked list of frame string records

Procedure InsaSelI
Insert a selection record after a given selection record that is on a linked list of selection records

1.3.4.4 Insert at the end of a list utilities

Procedure InseFsI Insert a frame string record at the end of a frame string recordlinked list

Procedure InseSelI
Insert a selection record at the end of a selection record linked list

1.3.5. NetLib

Module NetLib is effectively NetHandl, NetInsert, NetMakeDel, NetOption, NetStack and NetString. It is still used to keep older programs and modules compatible with the new division and to allow users to import only one module instead of six.

1.3.6. NetMakeDel

Module NetMakeDel impliments all the routines that make and delete records and save the records for future use when needed. This is basically a memory manager for frame string records, frame string15 records and selection records.

1.3.6.1 Initialization procedures

Procedure IniNetMakeDe
Initialize variables and pointers in NetMakeDel

Procedure IniFBody
Initialize only the body of frame record (not the frame header)

Procedure IniFP Initialize the entire frame record structure

1.3.6.2 Save a record utilities

Procedure SavFsP Save a frame string record on a save list. This routine puts an unused frame string record on the save frame string record linked list

Procedure SavSelP
Save a frame selection record on the save list

Procedure SavFP Save a frame record structure on the save list

1.3.6.3 Release memory utilities

Procedure RelFsP Release a frame string linked list to the save list. This routine will put every frame string record pointed to by a frame string pointer on the save frame string record linked list.

Procedure RelSelP Put a linked list of selection record structures on the save list

Procedure RelFBody
Release the contents of the body of a frame record structure. This routine will release the entire substructure of a frame except for the frame header information.

Procedure RelF Release the entire contents of a frame record. This routine will release the entire contents of a frame record structure to the various save record linked lists that exist.

Procedure RelFHP Release a frame header pointer and put it on the free list

Procedure RelFP Release a linked list of frame pointers and put it on the save list.

1.3.6.4 Clear the contents of a record utilities

Procedure ClrFBody
Clear only the body of the frame record (Don't change header). This routine will release the entire substructure of a frame record structure to the various save lists and initialize all the pointers to nil except for the frame header information.

Procedure ClrFP Clear the entire frame record (release all substructure)

1.3.6.5 Create utilities

The create utilities will attempt to get the appropriate record structure from the appropriate save list. If the save list is empty then a new structure is created dynamically.

Function CrFsP Create a new frame string record strcture

Function CrFP Create a new frame record structure

Function CrSelF Create a new selection record structure

1.3.6.6 Delete utilities

The delete utilities will delete the appropriate record from the linked list t that is currently a part of. If the record is the top of the linked list the second record will automatically be made the top of the list.

Procedure DelFsl Delete a frame string record from a linked list

Procedure DelSell Delete a selection record from a selection record linked list

Procedure SetMrkSel
Set the mark (space or minus) in a selection

1.3.7. NetOption

Module NetCption impliments the routines that manipulate options or local pads within a frame. They include finding, inserting and creating.

Function GOptF Get an option with a given selection character from a frames option list

Procedure InsOptF Insert a option in the option list of a frame

Function GPadF Get a pad with a given selection character from a frame's local pad list

Procedure InsPadF
Insert a selection in the pad list of a frame

Function CvStrSelTxt
Convert a pascal string to selection text

Function CrOptF Create a new option in the frame

Function CrPadF Create a new pad in the frame

Function GNewOpt
Find where next available option on a frame should go

1.3.8. NetPERQCodes

1.3.9. NetStack

Module NetStack impliments the stack operations push, pop and read for ZOG frames. A frame stack is a mechanism to remember what frames have been visited in the past and allow them to be visited again. It is basically a linkeked list of frame string15 records that hold the frame id of all frames "pushed" on the stack.

Procedure InitFstk Initialize fields of frame buffer stack Fstk

Procedure IniNetStack
Initialize variables and pointers in the module NetStack

Procedure RdFstk Read a frame from the net file into the top of the frame buffer stack

Procedure PshFstk Preserve the current top frame in a frame buffer stack

Procedure PopFstk
Restore last preserved frame to the top of frame stack Fstk

Procedure RdfstkX Read a frame from the net file into the frame buffer stack X

Procedure PshFstkX
Preserve the current top frame in a frame buffer stack X

Procedure PopFstkX
Restore last preserved frame to the top of frame stack Fstk X

1.3.10. NetString

Module NetString handles all of the string handling routines for the ZOG system.

1.3.10.1 Convert utilities

Function CvUcLc Convert a character to a lower case alphabetic

Function CvLcUc Convert a character to an upper case alphabetic

Procedure ConvLower
Convert a string to all lower case characters

Function AnyPos Find the pition of a mask in a string

Function Narrow Converta long integer to a normal integer

Function Widen Convert a integer to a long integer

1.3.10.2 Character string manipulation

Procedure Strip Strip carrage returns, line feeds and blanks from the front and back of a Pascal string

1.3.10.3 Frame string utilities

Function TFsNull Test to see if a frame string ponter is nil

Function GFs15P Get a frame string15 pointer that matches a mask

1.3.10.4 String equality utilities

Function TEqStrCase
Test to see if two strings are equal (case sensitive)

Function TEqStrSub
Test to see if one string is a substring of another

1.3.10.5 String-long conversion

Function RoundLong
Convert a real number into a long integer rounded to the nearest whole number

Function TruncLong
Convert a real number to a long integer truncated to the nearest whole number

Function FloatLong
Convert a long integer to a real number

Procedure CvRealStr
Convert a real number into a character string

Function CvStrReal
Convert string to a real number

1.3.10.6 Time and date

Procedure CvTimIntStr
Convert a time integer in milliseconds since midnight into a pascal string in the form HH:MM:SS

Procedure CvMonIntStr
Convert a month integer to a pascal string

Procedure CvDatIntStr
Convert a date integer to a pascal string of the form DD MM YY

1.3.10.7 General utilities

Function TFidValid Test to see if a frame id is valid

Procedure PrsFid Parse a frame id into its subnet name and relative frame number

1.4. Net interface module

1.4.1. Module NetHandl

Provides upper-level procedures for accessing the NetServer modules. The NetServer modules provide read and write access to all the subnets and frames within the network of PERQs linked together with the EtherNet. Since that access is at a lower level than the procedures contained herein, the details of local versus remote access are completely hidden from the user or agent writer.

1.5. Screen interface

1.5.1. Module IncDisp

Module IncDisp is used for monitoring the state of a frame designated to be automatically updated as it is changed, presumably by some remote user(s). When a frame is re-displayed via this mechanism, the changes that have been made are highlighted in reverse video. The polling for this mechanism which determines when a frame re-display might be necessary is handled in ZSel.Return and the ZCanvas routine RdTKeyZOG.

InitIncDisp Create scratch record and initialize incremental display (local boolean, IncDispSig) to off.

SetIncDisp Given an update timing interval, set local variable IncDispTime and IncDispSig accordingly, to turn incremental display on or off, as requested.

UpdateIncDisp Redisplay frame with highlighted changes if frame has changed and if display timing interval has expired.

SavIncDisp Save the frame ID of the frame displayed in the current window and mark it as not having changed.

1.5.2. ZCanvas

ZCanvas provides the lowest level screen display routines.

1.5.2.1 Canvas (window) and pointer routines

InitZCanvas	initialize the all canvas variables
ChangeCanvas	Go to another window
TitleCanvas	Insert Title string at top of current window
ClearCanvas	Clear the current window
SetCanvas	Read in internal window variables
ResetCanvas	Clear the window and reset the screen
SetCanvPtr	Select an image for the mouse pointer
SetCanvFunc	Select black/white background
SetPtrCh	Move mouse pointer and alternate cursor, if applicable.
SetPosCh	Set the position of the mouse pointer
GetPosCh	Get the current position of the mouse pointer

1.5.2.2 cursor controlling routines

CursorOn	Turn on the (character input, i.e., underscore) cursor
CursorOff	Turn off the cursor
isCursorOn	Return the current state of the cursor
SetAltCursor	Select an image for the alternate cursor
DispAltCursor	Display a new alternate cursor at specified location
IsAltCursor	Return the current state of the alternate cursor
LineCanvas	Draw a line between specified points
BoxCanvas	Draw a box with specified corners
SetCursorCh	Set the cursor character

1.5.2.3 Character (and mouse) Input and Output

Rd[T]Canvas	Get next character/mouse input from user
Rd[T]KeyEv	Get next character input; stay in current window
Rd[T]KeyZOG	Get next character input; can change windows
Rd[T]Kbd	Get a character from keyboard
RdKbdCond	Return character or null
RdCanv	Detect input event and return it in a canvEv record

IsMouseEv	Return true if Canvas event was mouse button click
SetChFunc	Select Replace, OR, XOR, etc. character display function
GetChFunc	Return the current character display function
PutCh	Output a character to screen and external terminal
PutStr	Output a string
PutStrLn	Output a string followed by a cr/lf
PutSubStr	Output a substring
BEEPCanvas	Cause the terminal speaker to emit a short "beep"

1.5.3. Module ZCanvUtils

SetZOGCanvas	Initialize 5 ZOG Frame Canvases and record structures
IniCanvPtr	Read in Mouse pointer images from :zognet>zog.animate
CharToAbs	Convert normal x-y character coordinates to screen pixel (absolute) coordinates
AbsToChar	Convert pixel (absolute) coordinate points to character x-y cordinates, in the proper window
ScreenLine	Draw a line on screen connecting two absolute coordinates
DrawBox	Draw a box on screen with opposite corners given in absolute (pixel) coordinates

1.5.4. Module ZDisplay

The ZDisplay provides complex frame and window display utilities.

Clear	Clear the current window
DspPos	Position the cursor to specified location in the frame display
DspEnd	Position the cursor to (UsrDispLine,1)
ClrEOLn	Clear from current position to end-of-line
ClrLine	Clear the entire current line
DspF	Clear the current window and display specified frame
DspF1	Display specified frame without clearing
DspSelF	Display Selection records
DspLPads	Display Local Pads records
DGPads	Display Global Pads from the specified GPads frame
DspFsP	Display a Frame String record (i.e., item text)
DspStar	Mark the selection with an asterisk
DspCxt	Display a context string ('edit','second',etc.) to the left of the frame id (upper RH corner)

1.5.5. Module ZIO

The ZIO module controls message display on the user display line of frame windows, and message handling to external devices via the RS-232 port.

1.5.6. Module ZUser

The ZUser module contains procedures which control the messages sent to the User Display Window, and the windows themselves.

WrUsrDsp Write a single character to the User Display Window.

ClrUsrDsp Clear the User Display Line and Window, clearing all the user display data structures.

DspUsrDsp Change windows to the Full Screen User Display window, and display the last screen-full of error messages.

DspFUsrDsp Display the last User Display Window-full of (error) messages in the (small) User Display Window.

InitUser Allocate memory for user display routines

InitUsrDsp Initialize all the variables for the User Display

1.5.7. Module ZWind

The ZWind Module performs the task of managing the ZOG frames in their separate windows with the following EXPORTed procedures.

XChange Change current window (to the other window) (†aw action)

SetWind Set the name of the frame, global pads frame and the selection character used to get there into current window record

DspWind (Re)Display the frame in the current window; this is also used to leave the full-screen user display

RdFWind Read a frame into the current window frame record; in general, a backup stack entry should be pushed before this is called

OpnWind Open the frame in the current window (lock the record and read it in) (presumably for subsequent modification)

InitWind initialize the vatiables in ZWind

ReInitWind Reinitialize window records for newly logged-in user

1.6. Action Processing Modules

1.6.1. ZAAction

1.6.1.1 Window Utilities

SetUserLine	Set the User Display line number
ExpandBig	Expand the current frame to a big Frame
ShrinkBig	Change the current big frame to a normal two window display
DispFr	Redisplay frame in current window

1.6.1.2 Subnet Utilities

TopOfSubnet	Go to the top frame in the current subnet
SetInitFr	Set the name stored in top.frame. Top.frame used to be the t top frame displayed at login
SetFrFile	Write a frameid to a specified file
GoToFrame	go to a specified frame
TopFrOfNet	Go to the frame listed in top.frame
Pop	Pop a frame from the current windows backup stack
CntlASel	main selection routine for processing ↑A actions

1.6.2. Module ZAction:

InitAction	Initializes data structures for Control-D actions and for the Video disk modules
XAction	Causes the given action command to be executed by dispatching it to the appropriate Module Procedure
ProActStr	Parses the given string into action commands, then passes the command to XAction for execution
ProAction	Removes individual strings from an action string record, then passes these to ProActStr for eventual execution
GAction	Given a control character input by the user, obtains the rest of the action command and optional action arguments, then causes them to be executed

1.6.3. Module ZActUtils:

This module contains a number of utilities for the action processing modules.

CvFid	Returns valid frame id from user-input string
CvSid	Returns valid Subnet Id from user-input string
CvFile	Returns a filename from user-input string
GFidUsr	Issues prompt to user for frame id, and returns frame id
GSidUsr	Issues prompt to user for Subnet Id, and returns subnet id
GFileUsr	Issues prompt to user for filename, and returns it

TExitAction	Returns true if current frame has an exit action
GetAction	Returns action string record from selection pointer
DoAction	Causes the given action string to be executed

1.6.4. ZBAction

- Comment : Writes the contents of a frame comment area to the user display
- ClsFile : Close a file
- PosCursor : Position the cursor to a specified row and column
- OpnInFile : Open a file for input (read only)
- OpnOutFile : Open a file for output (write only)
- PrintChar : Print a char in front of a given option
- ChangeTerm : Change the type of terminal that ZOG will send its output to
- CntlBSel : main routine to process ↑B actions

1.6.5. ZDAction

1.6.5.1 InitDAct : Initialize the variable for ZDaction

1.6.5.2 A - I command procedures : procedures for ↑D actions

AddOwner	Add an owner to the current frame
ClearSn	Clear a given subnet
CrFrame	Create a new frame with a given frameid
CrSubnet	Create a new subnet
EditFrame	Invoke the Zog editor (Zed)
EraseFrame	Erase a given frame
FProtect	Set the protection for a frame
Info	Write the frame header info to the user display and highlight any differences between the current frame and the old frame

1.6.5.3 J - Z commmand procedures : procedure for ↑D actions

- Play : Play back a script
- PbRecord : Record a script
- ShowStats : Display the status of a given perq in the user display

- ShowAllStats :Display the status of all the Perqs in the user display
- SlotEditFrame : Invoke the slot editor (Sled)
- WrFrBh : Write a frame in BH format
- WrSNetBh : Write a subnet in BH format

1.6.5.4 CntlDSel : main routine to process ↑D actions

1.6.6. ZEAction

- PrintFile : Send a file to the print server
- ZScreenDump : Send a copy of the current screen image to the print server
- CntlESel : main routine to process ↑E actions

1.7. Exernal Device and Utility I/O

1.7.1. UEI

1.7.1.1 Utilities

- UEIActivate : Activate the Universal External Interface controller
- UEIDeActivate : DeActivate the Universal External Interface controller
- UEIEcho : Display/Don't display command
- UEIBeginStack : Clear the video stack
- UEIEndStack : Termainate the command stack
- UEIReset : Reset the Universal External Interface controller

1.7.1.2 Video commands

- Disk control commands
- Auxillary commands
- Programming Commands

1.7.1.3 Utility commands

1.7.2. ZBHIO

1.7.3. ZVideo

1.8. Polling Routines for Statistics or AirPlan

- ZPollSnap
- ZPoll
- ZPollProc
- ZPollAir

1.9. Statistics Gathering

- StatsDefs
- StatsLib
- ZPutStats
- ZSnapShot
- ZStats

1.10. Miscellaneous Utilities

- ZDump
- ZError
- ZLogFile
- ZTrace

2. ZOG Netserver Modules

These modules provide access to subnets and frames anywhere within the ZOG net, i.e., on both the local PERQ and on any other PERQ linked to the local PERQ by the EtherNet. They are accessed from the basic ZOG system via procedures in module NetHandl, which accesses the rest through ZAccessProcs.

- E10Types

- ZAccessProcs
- NetServ
- ZNet
- ZEInt
- ZNetServer
- ZNetProcs
- ZOGMsg
- ZOGMsgDefs
- ZOGNetServer

2.1. E10Types

This module contains all valid Ethernet type fields used by PERQ software. This file is meant to be used as an include file.

2.2. ZAccessProcs

Most ZAccessProcs routines provide an interface between higher level routines (i.e. those in NetHandl) which make requests to access or modify a frame or subnet, and lower level routines (i.e. those in NetServ and ZNet) which perform the actual accessing of the frames and subnets. The other routines in ZAccessProcs, the login/logout and utility routines, are themselves low level routines.

All routines in Module ZAccessProcs return an integer value declared as type GeneralReturn in Module NetDefs. This integer value represents either Success, or a value for some type of failure signal. These signals and the integer value for success can be found declared as constants in Module NetDefs.

The routines in Module ZAccessProcs have been broken into five categories to coincide with their calling routines in Module NetHandl:

- Frame Access Routines
- Frame Modification Routines
- Subnet Access Routines
- Utility Routines

- Zog and Agent, Login/Logout Routines

2.2.1. Frame Access Routines

These routines provide an interface between higher level and lower level frame access (view, create, delete) routines. Although doing very different things, they use very much the same method in locating a subnet or frame. This is detailed in Functions ReadFrame, ReadHeader, OpenFrame, and CloseFrame. The functions call the appropriate functions in either Module NetServ, for local frames and subnets, or routines in Module ZNet for accessing frames and subnets on a remote machine. These Functions will return success to the calling routine in Module NetHandl if successful.

2.2.1.1 Function ReadFrame

ReadFrame will read a frame locally if possible. Otherwise, it scans the local server database to find any primary or secondary node which is up, and sends a request to that node. If none are listed as up in the local server database, it probes each of the primary and secondary nodes to find one which is up. If it finds one, the server data base is updated and the request is forwarded. The routine returns success if it is able to read the frame specified. Otherwise, it returns any of a number of failure signals. it proceeds as follows:

2.2.1.1.1 Calls Function ZAccessProcs.CheckServer to make sure the request for a frame comes from a logged in user.

2.2.1.1.2 Calls Function ZogNetServer.GetSnRecord which hashes into the local subnet database for the record. If the record is not represented locally, the routine searches in the subnet index of the master node. If the record is found, it is added to the local subnet database. An attempt is made to open the file if it is on the local disk.

2.2.1.1.3 If the Current node has the primary copy of this subnet, then read the frame thru a call to Function NetServ.RdF-.

- If the Primary copy of the subnet is on a remote machine, call Function ZNet.ZReadFrame to read the frame from a remote machine. If ZReadFrame returns unsuccessfully, call Function ZogNetServer.Probe to see if the Primary node is actually up. If successful, call Function ZNet.ZReadFrame again, to read the frame from the remote machine.

- If there is no success reading from the primary node, ReadFrame next checks to see if the current node has a secondary copy of the subnet. If it does, ReadFrame calls Function NetServ.RdF- to read. Otherwise, it checks the remainder of the secondary nodes in the same fashion as it did for the primary node, looking for one that is up, so that a call to Function Znet.ZReadFrame can read the frame from a remote machine.

- In the event that the primary and secondary nodes are all not up, then ReadFrame returns the signal - SigFrUnavailable.

2.2.1.2 Call ZAccessProcs.ReadHeader

After initializing the global variables needed to access the network (and returning an error if something was amiss), ReadHeader will attempt to locate the subnet that contains the frame. Then, it will try to read the frame header:

2.2.1.2.1 If the subnet does not exist, return an error.

2.2.1.2.2 If the current machine contains the primary copy of the subnet, call NetServ.RdFH - to obtain the header and exit

2.2.1.2.2.1 NetServ.RdFH -

- Test to see if the subnet exists on the local machine. If not, there is an inconsistency in the subnet indexes. Return an error.
- Calculate the page number of the first frame. There are 10 pages/frame so this is easy.
- Turn off ethernet interrupts before reading from the disk.
- Read in the page header.
- Turn interrupts back on.
- If the first byte is null, the frame does not exist. Return an error.
- Parse the frame into the frame record.
- If the frame is protected, return an error.
- Return success.

2.2.1.2.3 If the remote host is listed as being 'up', call ZNet.ZReadHeader to read it from a remote machine and exit if successful,

2.2.1.2.3.1 ZNet.ZReadHeader

Note: The following algorithm is repeated a maximum of 'MaxRetries' (3) times.

- Recast the ZOG message buffers as ReadHeader message buffers.
- Load up the parameters of the message buffers.
- Call ZOGMsg.SndRcvRecord to send the message over the Ethernet.
- If the returned status was not success, repeat.

- If the message buffer contains an error, repeat.
- Call ZOGMsg.ReceiveBuffer to get the header page.
- Calls Function ZogMsg.ReceiveBuffer to get the frame body.
- If that failed, repeat the initial request.

2.2.1.2.3.1.1 Recast the ZOG message buffers as ReadHeader message buffers.

2.2.1.2.3.1.2 Load up the parameters of the message buffers.

2.2.1.2.3.1.3 Call ZOGMsg.SndRcvRecord to send the message over the Ethernet.

SndRcvRecord does a synchronous send/receive pair between two machines connected by the EtherNet. The routine sends a message across the EtherNet and waits (with a timeout) for a reply. Errors are returned accordingly.

The messages generated by SndRcvRecord cause EtherNet exceptions to be raised on the local (sending) machine and target (receiving) machine, and these then raise the 'E10ReceiveDone' exception, locally and within ZOG. The local E10RecieveDone handler of SndRcvRecord handles the acknowlegement and reply of the target machine to the local machine. The E10RecieveDone handler at the ZOG system (in Module Zog) level handles the receiving of the request of the sending machine and processing it.

2.2.1.2.3.1.3.1 ZOG.E10ReceiveDone

The EtherNet exception handler in ZOG is invoked when a remote machine sends the current machine a message. This exception handler contains nested handlers to protect ZOG from dying when additional messages are received while ZOG is processing ethernet messages.

2.2.1.2.3.1.3.1.1 Change the mouse image to the hollow arrow. This is purely cosmetic.

2.2.1.2.3.1.3.1.2 Call ZOGMsg.HandleMsg to get the message buffers. If an error occurs, ignore the message. The other machine will resend it if it is important enough.

2.2.1.2.3.1.3.1.2.1 Function ZOGMsg.HandleMsg

This is a boolean function that returns true if there is a valid ZOG request. The message is taken from EtherNet packet form and put into a ZOG message record that must be handled. It returns false for those messages not of the ZOG record protocol. It does the following:

- Check to see if there is a legal ZOG record message.

- If the message received is a request for a probe, and machine names match, then send a probe reply. Thus, the probe is handled right here and HandleMsg returns false.

- If the message is an acknowledgement then return a value of false. These are ignored at this level to avoid infinite loops that can occur with two machines that get out of synchronization and begin sending Ack messages back and forth.

- Otherwise, send an acknowledgement to the sending machine and transfer the received message from the buffer to a ZogMsgPTyp and return true.

2.2.1.2.3.1.3.1.3 Pass the message buffers to ZNetServer.ZNetServer for processing.

Module ZNetServer is the counterpart of Module NetHandl on the remote machine. It invokes the local routines which will return the necessary data. It is simply a case statement which uses the input message id to determine which routine should be called. In this example, it will call Procedure ZNetServer.XZReadHeader.

2.2.1.2.3.1.3.1.3.1 Procedure ZNetServer.XZReadHeader.

XZReadHeader is the equivalent, on the remote machine, to Procedure NetHandl.RdFH on the local machine. Its method is very differnt from that of Procedure NetHandl.RdFH because of the fact that it must perform its task on a remote machine. Notice, however, that both call Procedure NetServ.RdFH_ to do the low level reading of the frame. It does the following:

2.2.1.2.3.1.3.1.3.1.1 Recasts variables local to the Procedure as ethernet request and reply types.

2.2.1.2.3.1.3.1.3.1.2 Calls Function ZNetProcs.ZReadHeader to load the header block into the message buffer

ZNetProcs is the counterpart to ZAccessProcs and handles access on a remote machine. For details on NetServ.RdFH_ see p. 22.

- Verifies that the subnet exists, via a call to Function ZogNetServer.Chk· SnRecord. ChkSnRecord is very similar to GetSnRecord, except that the subnet information passed along with the request is assumed to be correct. This eliminates the need to request it from the MasterNodes Subnet Index. So only the local subnet index needs to be examined to make sure that the information is correct.

- Otherwise call NetServ.RdFH – to load the buffer with the frame header block.

2.2.1.2.3.1.3.1.3.1.3 Calls Function ZogMsg.SendRecord to send a reply to the sending machine.

Sends a record to a remote machine and waits for an acknowledgement of receipt of the record.

2.2.1.2.3.1.3.1.3.1.3.1 Sets addresses to be correct, in various records, so that the record can be received on the remote machine.

2.2.1.2.3.1.3.1.3.1.3.2 Resend Loop

At this point the SendRecord enters a loop to send a request to the other machine, saying, "Well, Go ahead". The loop will attempt to send the request a maximum of NumberResends times (5). To send the request, first the ethernet interrupts are turned off. Next, a call is made to Procedure Ether10IO.E10WIO which starts an EtherNet I/O operation and waits for it to complete. In this case, information is being sent, so E10WIO makes sure the information is sent over the EtherNet.

2.2.1.2.3.1.3.1.3.1.3.3 If an error is detected in sending the message, then exit SendRecord. Otherwise, set the EtherNet Handler State to indicate that the local machine is waiting for the acknowledgement from the remote machine (SWaitAck) and turn on the EtherNet interrupts.

2.2.1.2.3.1.3.1.3.1.3.4 Got Acknowledgement Time-Controlled Loop

If the acknowledgement is received by the machine sending the message, an interrupt is generated, causing an exception to be raised by the EtherNet MicroCode, thus invoking the local Handler E10ReceiveDone. E10ReceiveDone sees that the EtherNet Handler State indicates that the local machine is waiting for an acknowledgement (SwaitAck), and signals acknowledgement by assigning the EtherNet Handler State to be that of 'Got the Acknowledgement' (SGotAck). If the acknowledgement is received, exit SendRecord.

2.2.1.2.3.1.3.1.3.1.3.5 If after five attempts no acknowledgement is received from the remote machine, then exit SendRecord with an error.

2.2.1.2.3.1.3.1.3.1.4 Calls Function ZogMsg.SendBuffer to send the actual frame header.

2.2.1.2.3.1.3.1.3.1.4.1 If PgCnt = 0 then exit SendBuffer successfully. The buffer is empty, and nothing is sent.

2.2.1.2.3.1.3.1.3.1.4.2 Set the Ethernet Handler State to indicate that this machine would like to go ahead and send a buffer (SWaitGo).

2.2.1.2.3.1.3.1.3.1.4.3 Wait for Go Ahead Time-Controlled Loop

At this point a time controlled loop is entered, and its purpose is to wait for an interrupt which indicates that it is all right to send the first buffer. If the interrupt occurs, the local handler E10ReceiveDone is invoked and acknowledgement is sent to the remote machine. If this acknowledgement is sent successfully, the EtherNet Handler State is set to indicate 'Go Ahead and Send the First Buffer' (SSendFirst). Otherwise, the handler is exited, leaving the Ethernet Handler State in the original state.

2.2.1.2.3.1.3.1.3.1.4.4 Set up records with correct addresses to send first buffer.

2.2.1.2.3.1.3.1.3.1.4.5 Resend Loop

- Assumes initially that only one page is being sent and puts that page into the buffer to be sent.
- Checks to see if there is more than one page to transfer. If so, sets the buffer page size to two and puts the second page in the buffer.
- Turns the Ethernet interrupts off and sends the Zog Buffer Packet with a call to Procedure Ether10IO.E10WIO. If sent successfully, sets the EtherNet Handler State to be that of 'Waiting for a Buffer-Received Acknowledgement' (SWaitBufAck) and turns Ethernet interrupts on. Otherwise, exits SendBuffer with an error.

2.2.1.2.3.1.3.1.3.1.4.6 Buffer Received Acknowledgement Time-Controlled Loop

At this point, again another time-controlled loop is entered. This time it is waiting for an interrupt indicating that the buffer was sent. If that interrupt occurs, the local handler E10ReceiveDone is invoked. It first examines the acknowledgement from the machine that received the buffer, for correctness. If the acknowledgement is correct and if all the information has been sent, the EtherNet Handler State is set to indicate that all has been sent (SSentAll) and the handler is exited. Otherwise, the handler attempts (only once) to send the next buffer itself, in the same fashion as SendBuffer.

2.2.1.2.3.1.3.1.3.1.4.7 If after five attempts the buffer has not been sent, then exit SendBuffer with an error.

2.2.1.2.3.1.3.1.3.1.5 Calls Function ZogMsg.SendBuffer to send the actual frame body.

2.2.1.2.3.1.3.1.4 Clean up the mouse image and anything else if necessary.

2.2.1.2.3.1.3.2 Function ZOGMsg.SndRcvRecord.

2.2.1.2.3.1.3.2.1 Establishes a time deadline for receipt of the reply record from the target machine.

2.2.1.2.3.1.3.2.2 Sets up several records to be sent over the ethernet, by the local machine. These are recast as ZogMsgPTyp's.

2.2.1.2.3.1.3.2.3 Resend Loop

At this point SndRcvRecord enters a loop to send the request for information to the target machine. The loop will attempt to send the request a maximum of NumberResends times (5). To send the request, first the ethernet interrupts are turned off. Next, a call is made to Procedure Ether10IO.E10WIO which starts an EtherNet I/O operation and waits for it to complete. In this case, information is being sent, so E10WIO makes sure the information is sent over the EtherNet.

2.2.1.2.3.1.3.2.4 If an error is detected in sending the message, then exit SndRcvRecord. Otherwise, set the EtherNet Handler State to indicate that the local machine is waiting for the acknowledgement from the remote machine and turn on the EtherNet interrupts.

2.2.1.2.3.1.3.2.5 Wait Reply Time-Controlled loop.

A time controlled loop is simply a loop that terminates after a certain period of time. This is here so that the local machine can wait for an acknowlegement from the target machine, saying that it has received the request for information.

If the acknowledgement is received by the local machine, an interrupt is generated, causing an exception to be raised by the EtherNet MicroCode, thus invoking the local Handler E10ReceiveDone. E10ReceiveDone sees that the EtherNet Handler State indicates that the local machine is waiting for an acknowledgement (SwaitSndAck), and signals acknowledgement by assigning the EtherNet Handler State to be that of waiting for a reply (SWaitReply). If the acknowledgement has been received, exit the resend loop. Otherwise, continue to attempt to send the message, up to five times. and exit with an error if an acknowledgement is never received.

2.2.1.2.3.1.3.2.6 Got Reply Time-Controlled Loop

When an acknowledgement is received, another time controlled loop is entered, waiting for an interrupt which will again invoke the local handler E10ReceiveDone. This time, the Ethernet Handler State is indicating that the local machine is waiting for a reply (SWaitReply). If the exception is raised, E10ReceiveDone sets up the acknowledgement packet and the packet of information to be recieved.

It then assigns the EtherNet Handler State to be that of 'Got the Reply' (SGotReply). At this point, SndRcvRecord will exit successfully.

2.2.1.2.3.1.3.2.7 TimeOut.

2.2.1.2.3.1.4 If the returned status was not success, repeat.

2.2.1.2.3.1.5 If the message buffer contains an error, repeat.

2.2.1.2.3.1.6 Call ZOGMsg.ReceiveBuffer to get header page

ReceiveBuffer waits for a buffer from a remote machine connected by the EthorNet.

2.2.1.2.3.1.6.1 Function ZogMsg.ReceiveBuffer.

2.2.1.2.3.1.6.1.1 If PgCnt = 0 then exit ReceiveBuffer successfully. The buffer is empty, and nothing is sent.

2.2.1.2.3.1.6.1.2 Sets up several records to send an acknowledgement to the remote machine. This will acknowledge the 'Go Ahead' message sent by the remote machine during its synchronized execution of Function ZogMsg.SendBuffer. In essence, the local machine is saying, 'Go ahead and send me the information I requested, I am ready to receive it'.

2.2.1.2.3.1.6.1.3 Resend Loop.

At this point ReceiveBuffer enters a loop to send the acknowledgement from the local machine (which originally requested the information) to the remote machine (the machine sending the information), telling it to send the information that was requested (Go Ahead). It will attempt to send the acknowledgement NumberResends times (5). Each time it attempts to send the acknowledgement, it will enter a time-controlled loop to see if the acknowledgement sent was received by the remote machine. If an error occurs in sending the acknowledgement, then control exits ReceiveBuffer with an error. Otherwise, the EtherNet Handler State is set to indicate that we are ready to receive the information (SWaitGoAck). We then enter the Go Ahead Time-controlled loop (mentioned above) to wait for an interrupt to begin receiving.

2.2.1.2.3.1.6.1.4 Go Ahead Time-Controlled Loop.

After the acknowledgement is sent, the Go Ahead Time-Controlled Loop begins. Here, the machine which requested the information is waiting for an interrupt so that it can begin receiving information. If the interrupt occurs, an exception is raised and the local handler E10ReceiveDone is invoked. Inside the handler, another buffer, acknowledging the receipt of the buffer, is prepared, so that it can be sent

when the buffer is received. Finally, the Ethernet Handler State is changed to indicate that the machine will be receiving information (SReceiving).

Note: It is conceivable that the acknowledgement could be sent so quickly that a second interrupt could be generated before entering this loop. This is not likely, but if it does occur, the buffer will have been received before this time-controlled loop, in the handler, and the ethernet handler state will have been updated to indicate that the machine has gotten all the information (SGotAll).

2.2.1.2.3.1.6.1.5 If the interrupt is not generated, another attempt is made at sending the Go Ahead Acknowledgement. As usual, there are five attempts.

2.2.1.2.3.1.6.1.6 Wait Receive Time-Contolled Loop.

At this point the receiving machine enters a time-controlled loop to begin waiting for an interrupt indicating that the information requested has arrived. If that interrupt occurs, an exception is raised and the local handler E10ReceiveDone is again invoked, with the Ethernet Handler State being SReceiving. Inside the handler, a check is made to see that the information comes from the correct source and that the page numbers are correct. Finally, after all this effort, the requested information is transfered from the sending machine to the receiving machine and the receiving machine makes an attempt to send an acknowledgement to the sending machine. If the acknowledgement is not sent, the handler doesn't worry because the sending machine will time out.

Note : If the information was received in the first Time-Controlled loop, then the Ethernet Handler State will reflect this and ReceiveBuffer will exit successfully.

2.2.1.2.3.1.6.1.7 If the interrupt was received, then the information requested will have been received in the handler and the EtherNet Handler State will indicate that all information has been received (SGotAll). In this case, ReceveBuffer exits successfully. Otherwise, ReceiveBuffer will timeout.

2.2.1.2.3.1.7 Calls Function ZogMsg.ReceiveBuffer to get the frame body.

2.2.1.2.3.1.8 If that failed, repeat the initial request.

2.2.1.2.4 Else, probe the remote machine. If it responds, send the request and read the header and exit if successful.

2.2.1.2.5 If the current machine has a backup copy, use NetServ.RdFH –

2.2.1.2.6 Else, try all of the other backup copies with ZNet.ZReadHeader.

2.2.1.2.7 Otherwise, give up and return an error.

2.2.1.3 Function ZNet.ZOpenFrame

ZOpenFrame is designed to open a frame on a remote machine. It will attempt to send the message three times to the remote machine before it gives up. The number three was chosen arbitrarily. It uses two pointer types, de-

2.2.1.3.1 Intializes Variables and increments retries to begin a makeshift loop using labels.

2.2.1.3.2 The variables, local to the module ZNet, OutMsgP and InMsgP, are recast as open frame request and reply pointer types, to be used as such.

2.2.1.3.3 Prepare the EtherNet request packet, through various assignments.

2.2.1.3.4 Function ZogMsg.SndRcvRecord

SndRcvRecord does a synchronous send/receive pair between two machines connected by the ethernet. The routine sends a message across the Ethernet and waits (with a timeout) for a reply. Errors are returned accordingly. (For details see p. 27)

The messages generated by SndRcvRecord cause ethernet exceptions to be raised on the local (sending) machine and target (recieving) machine, and these then raise the 'E10ReceiveDone' exception, locally and within ZOG. The local E10RecieveDone handler of SndRcvRecord handles the acknowlegement and reply of the target machine to the local machine. The E10RecieveDone handler at the ZOG system (in Module Zog) level handles the receiving of the request of the sending machine and processing it.

2.2.1.3.4.1 ZOG.E10ReceiveDone

The EtherNet exception handler in ZOG is invoked when a remote machine sends the current machine a message. This exception handler contains nested handlers to protect ZOG from dying when additional messages are received while ZOG is processing ethernet messages.

2.2.1.3.4.1.1 Change the mouse image to the hollow arrow. This is purely cosmetic.

2.2.1.3.4.1.2 Call ZOGMsg.HandleMsg to get the message buffers. If an error occurs, ignore the message. The other machine will resend it if it is important enough.

For details on the inner workings of ZOGMsg.HandleMsg see p. 23.

2.2.1.3.4.1.3 Function ZNetServer.ZNetServer.

Module ZNetServer is the counterpart of Module NetHandl on the remote machine. It invokes the local routines which will return the necessary data. It is simply a case statement which uses the input message id to determine which routine should be called. In this example, it will call Procedure ZNetServer.XZOpenFrame.

2.2.1.3.4.1.3.1 Procedure ZNetServer.XZOpenFrame.

XZOpenFrame is the equivalent, on the remote machine, to Procedure NetHandl.OpnF on the local machine. Its method is very differnt from that of Procedure NetHandl.OpnF because of the fact that it must perform its task on a remote machine. It is important to notice, that both will call Procedure NetServ.OpnF to do the low level reading of the frame.

2.2.1.3.4.1.3.1.1 Recast variables local to the Procedure to be ethernet request and reply types.

2.2.1.3.4.1.3.1.2 Function ZNetProcs.ZOpenFrame.

ZNetProcs is the counterpart to ZAccessProcs and handles access on a remote machine.

2.2.1.3.4.1.3.1.2.1 Verifies that the subnet exists, via a call to Function ZogNetServer.Chk- SnRecord. ChkSnRecord is very similar to GetSnRecord, except that the subnet information passed along with the request is assumed to be correct. This eliminates the need to request it from the MasterNodes Subnet Index, so only the local subnet index needs examined to make sure that the information is correct. ChkSnRecord also opens the subnet by inserting it into the local subnet index and making sure that the file exists.

2.2.1.3.4.1.3.1.2.2 Function NetServ.OpnF-.

This function will open a frame for modification,lock it from access by other users, and read the frame from the file.

- Calls Function NetServ.GOpnUser to see if the current user has another frame open. If

so, exit with an error.

- Calls Function BaseLib.TSidValid to check for a valid Subnet Id.
- Calls Function NetServ.GSnRec which obtains the subnet from the local subnet index. The subnet was put there during initalization or during the call to Function NetServ.OpnSn – from Function ZogNetServer. GetSnRecord.
- If no errors have occured to this point, ethernet interrupts are turned off, via Procedure ZEInt.EIntOff. Then the frame header is read into a buffer by a call to Procedure FileSystem.FSBlkRead, and the frame header buffer count is assigned (FHBCnt).
- Calls Function NetServ.GOpnFid to see if the current frame is already open. If so, the ethernet interrupts are turned back on, via a call to ZEInt.EIntOn, and this user cannot access that frame.
- Calls Function NetServ.CrOpnRec which adds another frame to the list of open frames (refered to as locking the frame).
- More

2.2.1.3.4.1.3.1.3 Calls Function ZogMsg.SendRecord to send a reply to the sending mchine.

For details on the inner workings of ZOGMsg.SendRecord see p. 24.

2.2.1.3.4.1.3.1.4 Calls Function ZogMsg.SendBuffer to send the actual frame header.

For details of the inner workings of ZOGMsg.SendBuffer see p. 25.

2.2.1.3.4.1.3.1.5 Calls Function ZogMsg.SendBuffer to send the actual Frame body.

2.2.1.3.4.1.4 Clean up the mouse image and anything else if necessary.

2.2.1.3.5 Calls Function ZogMsg.RecieveBuffer if the reply came back successfully to receive the frame header.

For details on the inner workings of ZOGMsg.ReceiveBuffer see p. 28.

2.2.1.3.6 Calls Function ZogMsg.ReceiveBuffer to receive the buffer containing the frame body.

2.2.1.4 Function CreateFrame : Allows the user to create ANY specified frame.

2.2.1.5 Function CreateNextFrame : Allows user to create the next frame in a subnet

2.2.1.6 Function ZAccessProcs.CloseFrame

CloseFrame handles the closing of the frame on the primary machine and closing on a secdondary machine with different routines. This will become evident in the description of CloseFrame which follows;

2.2.1.6.1 Calls Function ZAccesssProcs.CheckUser to make sure the user is currently logged in.

2.2.1.6.2 Calls Function ZogNetServer.GetSnLocal which hashes into the local subnet index for the correct subnet and returns true if found. As mentioned, it checks only the local subnet index because the frame should have been opened.

2.2.1.6.3 Checks the local servers table to make sure the primary node is up. If not listed as up, calls Function ZogNetServer.Probe to see if the primary node is actually up. If so, the local servers table is updated. If not, exit with an error.

2.2.1.6.4 Checks to see if all machines with secondary copies are up in the same manner as described above. If a secondary node is not up then set to false an entry in the array SecUpdate (An array repre- senting the status of of secondary machines).

2.2.1.6.4.1 If the current machine contains the primary copy of the frame then call Function NetServ.ClsF – to close the frame.

2.2.1.6.4.1.1 Function NetServ.ClsF –

ClsF_ is designed to write and close a frame of the primary copy of a subnet.

2.2.1.6.4.1.1.1 Calls FunctionNetServ.GOpnUser and assigns the value of the frame on the open record list to a variable local to ClsF – .

2.2.1.6.4.1.1.2 Calls Procedure NetServ.SetModFH to set modification information of the frame. Modification information includes a new version number, the user modifiying, date, time, and whether modification was performed by an agent.

2.2.1.6.4.1.1.3 Turn off EtherNet interrupts, via Procedure ZEInt.EIntOff. Calls Procedure NetServ.WrFH to write the frame header to a buffer. Turn Eth- erNet interrrupts back on, via Procedure ZEInt.EIntOn.

2.2.1.6.4.1.1.4 Writes the header page and body pages of the modified frame to a file, via Procedure FileSystem.FSBlkWrite. If the frame gets smaller, due to the modification, a page of zeroes is written to the file to terminate the frame body. If the frame became larger, calls Procedure NetSev. SetFileLen to update the file to the new number of pages in the frame body.

- Turns the EtherNet interrupts back on.
- Calls Procedure NetServ.ErOpnRec to remove the open frame record from the list of open frames, thus unlocking the frame.
- Lastly, prepare modification information to be added to the file Change. Log. The file Change.Log stores information about every frame that is modified.
- Add Modification information to file Change.Log, via Procedure BaseLib. AppStrFile. AppStrFile will append the string to the end of a file.

2.2.1.6.4.2 Otherwise, the primary copy resides on another machine and Function ZNet.ZCloseFrame must be used to write and close the frame on a remote machine.

2.2.1.6.4.2.1 Function ZNet.ZCloseFrame.

ZCloseFrame is designed to close a frame on a remote machine. It uses types, from Module ZogMsgDefs, ClsF0PTyp as a close frame request packet, and ClsF1PTyp and ClsF2PTyp as close frame reply packets.

2.2.1.6.4.2.1.1 Initializes the local variables for loop using labels.

2.2.1.6.4.2.1.2 Recasts (restructures) the outgoing message (outMsg) and ingoing message (InMsg) to frame request and reply packets.

2.2.1.6.4.2.1.3 Prepares the EtherNet Request packet.

2.2.1.6.4.2.1.4 Calls function ZogMsg.SndRcvRecord to send a request to close a frame on the remote machine and receive a reply to that request. If not successful, then start all over, step 1.

2.2.1.6.4.2.1.4.1 Calls function ZogMsg.SndRcvRecord to send a request to close a frame on the remote machine and receive a reply to that request. If not successful, then start all over, step 1.

2.2.1.6.4.2.1.4.1.1 Function ZogMsg.SndRcvRecord.

SndRcvRecord does a synchronous send/receive pair between two machines connected by the ethernet (for details see p. 27). The routine sends a message across the Ethernet and waits (with a timeout) for a reply. Errors are returned accordingly. The messages generated by SndRcvRecord cause ethernet exceptions to be raised on the local (sending) machine and target (recieving) machine, and these then raise the 'E10ReceiveDone' exception, locally and within ZOG. The local E10RecieveDone handler of SndRcvRecord handles the acknowlegement and reply of the target machine to the local machine. The E10RecieveDone handler at the ZOG system (in Module Zog) level handles the receiving of the request of the sending machine and processing it.

2.2.1.6.4.2.1.4.1.1.1 ZOG.E10ReceiveDone on the target machine receives the message.

The EtherNet exception handler in ZOG is invoked when a remote machine sends the current machine a message. This exception handler contains nested handlers to protect ZOG from dying when additional messages are received while ZOG is processing ethernet messages. For details on ZOGMsg.HandleMsg see p. 23.

2.2.1.6.4.2.1.4.1.1.1.1 Change the mouse image to the hollow arrow. This is purely cosmetic.

2.2.1.6.4.2.1.4.1.1.1.2 Call ZOGMsg.HandleMsg to get the message buffers. If an error occurs, ignore the message. The other machine will resend it if it is important enough.

2.2.1.6.4.2.1.4.1.1.1.3 Pass the message buffers to Function ZNetServer.ZNetServer for processing.

Module ZNetServer is the counterpart of Module NetHandl on the remote machine. It invokes the local routines which will return the necessary data. It is simply a case statement which uses the input message id to determine which routine should be called. In this example, it will call Procedure ZNetServer.XZCloseFrame.

Procedure ZNetServer.XZCloseFrame.. XZCloseFrame is the equivalent, on the remote machine, to Procedure NetHandl.ClsF on the local machine. Its method is very differnt from that of Procedure NetHandl.ClsF because of the fact that it must perform its task on a remote machine. It is important to notice, that both will call Procedure NetServ.ClsF, to do the low level writing and closing

of the frame.

Recast variables local to the Procedure to be ethernet request and reply types.

Calls Function ZogMsg.SendRecord to acknowledge receipt of the request to write and close a frame. If not successful, exit XZCloseFrame erroneously.. Sends a record to a remote machine and waits for an acknowledgement of receipt of the record. **Sets addresses to be correct, in various records, so that the record can be received on the remote machine.**

Resend Loop. At this point the SendRecord enters a loop to send a request to the other machine, saying, "Well, Go ahead". The loop will attempt to send the request a maximum of NumberResends times (5). To send the request, first the ethernet interrupts are turned off. Next, a call is made to Procedure Ether10IO.E10WIO which starts an EtherNet I/O operation and waits for it to complete. In this case, information is being sent, so E10WIO makes sure the information is sent over the EtherNet. **If an error is detected in sending the message, then exit SendRecord. Otherwise, set the EtherNet Handler State to indicate that the local machine is waiting for the acknowledgement from the remote machine (SWaitAck) and turn on the EtherNet interrupts.**

Got Acknowledgement Time-Controlled Loop. If the acknowledgement is received by the machine sending the message, an interrupt is generated, causing an exception to be raised by the EtherNet MicroCode, thus invoking the local Handler E10ReceiveDone. E10ReceiveDone sees that the EtherNet Handler State indicates that the local machine is waiting for an acknowledgement (SwaitAck), and signals acknowledgement by assigning the EtherNet Handler State to be that of 'Got the Acknowledgement' (SGotAck). If the acknowledgement is received, exit SendRecord. **If after five attempts no acknowledgement is received from the remote machine, then exit SendRecord with an error.**

Calls Function ZogMsg.ReceiveBuffer to receive the frame body to be written. Again, if unsuccessful, exit erroneously.. For details of the inner workings of ZOGMsg.ReceiveBuffer see p. 28.

Calls Function ZNetProcs.ZCloseFrame to write and close the frame on machine where the frame exists.. For details on NetServ.ClsF, see p. 33.

- Calls Function ZogNetServer.GetSnLocal, which checks the local subnet index for the subnet containing the frame. Since ZCloseFrame is called only from the machine containing the primary copy of the frame, this is merely a double check to make sure the

frame is indeed there.

- Calls Function NetServ.ClsF – to do the actual low level writing and clos- ing of the frame.

Calls Function ZogMsg.SendRecord to send an acknowledgement to the machine sending the frame body, indicating the frame body was received.

Calls Function ZogMsg.SendBuffer to send frame header information back to the sending machine. Again, if unsuccessful exit erroneously.. If PgCnt = 0 then exit SendBuffer successfully. The buffer is empty, and nothing is sent. Set the Ethernet Handler State to indicate that this machine would like to go ahead and send a buffer (SWaitGo).

Wait for Go Ahead Time-Controlled Loop. At this point a time controlled loop is entered, and its purpose is to wait for an interrupt which indicates that it is all right to send the first buffer. If the interrupt occurs, the local handler E10ReceiveDone is invoked and acknowledgement is sent to the remote machine. If this acknowledgement is sent successfully, the EtherNet Handler State is set to indicate 'Go Ahead and Send the First Buffer' (SSendFirst). Otherwise, the handler is exited, leaving the Ethernet Handler State in the original state. **Set up records with correct addresses to send first buffer.**

Resend Loop.

- Assumes initially that only one page is being sent and puts that page into the buffer to be sent.
- Checks to see if there is more than one page to transfer. If so, sets the buffer page size to two and puts the second page in the buffer.
- Turns the Ethernet interrupts off and sends the Zog Buffer Packet with a call to Procedure Ether10IO.E10WIO. If sent successfully, sets the EtherNet Handler State to be that of 'Waiting for a Buffer-Received Acknowledgement' (SWaitBufAck) and turns Ethernet interrupts on. Otherwise, exits SendBuffer with an error.

Buffer Received Acknowledgement Time-Controlled Loop. At this point, again another time-controlled loop is entered. This time it is waiting for an interrupt indicating that the buffer was sent. If that interrupt occurs, the local handler E10ReceiveDone is invoked. It first examines the acknowledgement from the machine that received the buffer, for correctness. If the acknowledgement is correct and if all the information has been sent, the EtherNet Handler State is set to indicate that all has been sent (SSentAll) and the handler is exited. Otherwise, the handler attempts (only once) to send the next buffer itself, in the same fashion as SendBuffer. **If after five attempts**

the buffer has not been sent, then exit SendBuffer with an error.

2.2.1.6.4.2.1.4.1.1.1.4 Clean up the mouse image and anything else if necessary.

2.2.1.6.4.2.1.5 If successful, call function ZogMsg.SendBuffer to send the frame body to the remote machine. If not successful, then start all over at step 1.

For detials on the inner workings of ZOGMsg.SendBuffer see p. 28.

2.2.1.6.4.2.1.6 If SendBuffer was successful then calls Function ZogMsg.ReceiveRecord to receive the acknowledgement sent by the machine which received the frame body. If unsuccessful, start all over from step 1.

2.2.1.6.4.2.1.6.1 Function ZogMsg.ReceiveRecord.

ReceiveRecord receives a record from another machine. It is used only in CloseFrame function of Zog, because there are more acknowledgements that the SndRcvRecord can handle in closing a frame.

2.2.1.6.4.2.1.6.1.1 Sets the address from where the message should be received.

2.2.1.6.4.2.1.6.1.2 Sets the EtherNet Handler State to indicate that the sending machine is waiting for a reply (SWaitRcv).

2.2.1.6.4.2.1.6.1.3 Got Reply Time-Controlled Loop.

At this point the procedure will enter a time controlled loop waiting for an interrupt which will again invoke the local handler E10ReceiveDone. This time, the Ethernet Handler State is indicating that the local machine is waiting for a reply (SWaitRev). If the exception is raised E10ReceiveDone assigns the EtherNet Handler State to be that of 'Got the Reply' (SGotRev). At this point, ReceiveRecord will exit successfully.

2.2.1.6.4.2.1.6.1.4 If the interrupt is never received, ReceiveRecord will time out.

2.2.1.6.4.2.1.7 If the acknowledgement is received, calls Function ZogMsg.ReceiveBuffer to obtain the updated frame header information of the frame that was closed.

For details on the inner workings of ZOGMsg.ReceiveBuffer see p. 25.

2.2.1.6.4.3 Following this, Each secondary copy of the frame is updated. If the current machine has a secondary copy then, calls Function NetServ.Update to write and close the secondary copy.

2.2.1.6.4.3.1 Function NetServ.Update

UpDate is called only in the event that the current machine has a sec- ondary copy of the subnet of the frame. It is very similar to Function NetServ.ClsF_ with the exception to the following two items;

- In the beginning, It must obtain its subnet information from the local subnet index instead of the list of open frames. This is because the primary copy has been written and closed, and the open frame record has been removed from the list of open records.
- Lastly, this information is not added to the file Change.Log, because it lists only what frames have been modified, not each individual frame and backup copy modified.

2.2.1.6.4.4 If the secondary copy belongs on a remote machine then calls Function ZNet.ZCloseFrame. Remember, a frame can have copies on as many machines as the creator of the subnet specified.

2.2.1.6.4.5 If there were no secondary update failures (all secondary machines were up), then CloseFrame was successful and exit.

2.2.1.6.4.6 Otherwise, calls the nested Procedure ZAccessProcs.CloseFrame.SavSecUp- date. This will store is file sec.update the frame number,subnet ID, version number, date, time, curusername and a list of machine names (server names) of those secondary updates which failed.

2.2.1.7 Function QuitFrame : Closes a frame, but will not write to frame.

2.2.1.8 Function EraseFrame : Deletes a frame.

2.2.2. Frame Modification Routines.

Frame Modification routines modify existing frames. These routines follow the same form as the Frame Access Routines, in terms of locating the subnet of a frame. That is, calling routines in Module NetServ for modifying a local frame and calling routines in Module ZNet to modify frames on a remote machine. One important difference here is that these routines are called via Agents, which means that these frames are already open when a frame modification routine is called. Thus, the subnet containing the frame will already be listed in the local subnet index. This results in a call to Function ZogNetServer.GetSnLocal instead of a call to Function ZogNetServer.GetSnRecord in the frame modification routines. Both return the same information, GetSnLocal simply does it with less work.

These routines will return success if successful.

- Function AddOwner - calls either Function NetServ.AddOwnF - (local) or Function ZNet.ZAddOwner (remote), to add new owner.
- Function RemoveOwner - calls either Function NetServ.RemOwnF - (local) or Function ZNet.ZRemoveOwner (remote), to remove owner.
- Function SetFrProtection - calls either Function NetServ.SetProt - (local) or Function ZNet.ZSetProtection, to set frame protection bits.

2.2.3. Subnet Access Routines.

Subnet Access Routines are not as similar in nature as frame access and frame modification routines. Some use routines in Module NetServ and Module ZNet, while others do not. A brief summary of each of the functions is given;

2.2.3.1 Function CreateSubnet

CreateSubnet will call Function ZogNetServer.EnterSubnet to update the MasterNodes Subnet index and update the subnet index file to reflect the addition of the subnet being created by the calling routine, Procedure Nethandl.CrSnSec. Physically, there are no new frames stored on disk, just the addition of the subnet name to the proper indexes.

CreateSubnet will select the primary node, unless it is unlisted in the net.servers database, then the primary node becomes the master machine. In dealing with secondary nodes, CreateSubnet assigns values to an array representing those machines those machines to recieve backup copies. This in- formation is obtained from the file Sec.Default, during initialization and stored in a global array which is imported from Module ZogNetServer.

2.2.3.2 Function ClearSubnet : Clears (deletes) a subnet.

2.2.3.3 Function IsSubnetDefined : Checks to see if a subnet is defined in local or Master Subnet Index.

2.2.3.4 Function GetHiSubnet : Returns to the calling routine, the highest frame number in the subnet.

2.2.3.5 Function GetNextSubnet : Generates the next subnet in the master node subnet index.

2.2.4. Utility Routines

These utility routines are called by the Module NetHandl Utility routines and are summarized below.

Function GetCurNode

Returns to the calling routine the Current node (machine number) and Current machine name (i.e. mach1).

Function GetCurUser

Returns to the calling routine the current user Name (i.e. rch).

Function GetNodeName

Returns a machine name to calling routine.

Function GetAgentFlag

Returns success to the calling routine if an agent is currently being run.

2.2.5. Zog and Agent, Login/Logout Routines

These routines are not currently in use. They were designed to run under Spice (the forerunner to ZOG), to allow for spawning.

2.2.5.1 Function ZogLogin

Inserts another user into the the table of current zog users, represented by the array LoggedIn, which is declared in Module ZogNetServer and Exported to Module ZAccessProcs. This function also increments the variable MaxZOGPorts, which is also declared in Module ZogNetServer and which is a count of the number of currently logged in users.

2.2.5.2 NetString - String <=> Numeric Conversion Utilities

These routines make use of the PERQ PASCAL extensions such as Trunc, Float and Round and Stretch to handle most of the work.

Function RoundLong

Converts a real number to a rounded long integer

Function TruncLong

Converts a real number to a truncated long integer

Function FloatLong

Converts a long integer to a truncated real number

Procedure CvRealStr

Converts a real number to a character string

Function CvStrReal

Converts a character string to a real number

2.2.5.3 Function AgentLogIn : Adds a new entry into array LoggedIN and sets the AgentFlag field of this array entry to true indicating that this spawned process is an agent

2.3. NetServ

Module NetServ contains lower level routines which access the frame or subnet, except for the actual I/O, which is done at some still lower level in Module FileSystem, and routines which aid in this process. These routines are called by Module ZAccessProcs (for local requests) or Module ZNetProcs (for remote requests).

The routines are broken down into categories to coincide with their higher level routines in ZAccessProcs and ZNetProcs:

2.3.1. Frame Access Routines

These routines can, for the most part, be related to their Module NetHandl relatives by looking at the routine names. All have the same routine names followed by an underscore, with the exception of Function UpdateF which has no corresponding Module NetHandl routine. All will return an integer (Gen- eralReturn) value, that of the constant 'success', if successful.

Function RdF – Reads a frame from a file into a buffer

Function RdFH – Reads a frame header from a file into a buffer

Function OpnF – Opens a frame for writing, locks it from access by other users, and reads the frame into a buffer

Function CrFr – Creates a specified relative frame

Function CrF – Creates the next frame in the subnet

Function ErF – Deletes a frame from a subnet by writing pages of zeroes

Function ClsF – Writes and Closes a modified frame on a primary node

Function UpDateF Writes and Closes a modified frame on a secondary node.

2.3.2. Frame Modification Routines

These routines do the actual work of modifying existing frames. In the same way as the other Net Server routines, these can be recognized by their corresponding Module NetHandl Routine Names, with the addition of the underscore character following the NetHandl routine name. All will return the integer value of 'success' (type GeneralReturn from Module NetDefs), if successful.

2.3.3. Subnet Access Routines

2.3.4. Utility routines

These utility routines are called only by the other routines in Module NetServ. They can be broken down into the following areas:

2.3.4.1 Subnet Record Utilities

These utility routines work with subnet records (SnRecPTyp) in the local subnet index (SnTable).

Function HashSid Returns an integer value representing the hashed value of the subnet being sought in the local subnet index.

Function CrSnRec Creates a local subnet record and inserts it into the local subnet index. Returns a pointer to the new subnet record.

2.3.4.2 Open Frame Record Utilities

These routines deal with the list of open records, which is maintained and serves to lock other users from trying to modify a frame already opened. The open record list consists of OpnRecPTyp's. Type OpnRecPtyp, is Private to Module Netserv. Thus each machine has its own list of frames that have been opened locally or from a remote machine.

Procedure InsOpnRec

Inserts an open record at the beginning of the list of open records. It is called by another utility in this section, Function CrOpnRec.

Procedure DelOpnRec

Deletes an open record from the open record list. It is called by another Utility in this section, Function ErOpnRec.

Function CrOpnRec

Creates a new open record for the open record list.

Procedure ErOpnRec

Deletes an open record from the open records list. The deleted node is not disposed of, but saved as part of ZOG's own garbage collection mechanism.

Function GOpnUser

Searches the open record list for a frame, to see if it is already open. Returns a pointer to the open record frame.

Function GOpnFid Same purpose as GOpnUser, only uses different fields on which to search.

2.3.4.3 Subnet Utility

This function is called by Function NetServ.ClsF, if a modified frame is larger than before it was modified.

Procedure SetFileLength

Sets the length of a file to a given number of pages.

2.3.4.4 Frame Handling Routines

2.3.4.4.1 Procedure WrFH : Writes a Frame Header in record form into a buffer in ZBH form. Uses several nested procedures to write strings to the buffer.

An individual frame is written out to disk in a modified "BH" format, called ZBH format. This format was developed at CMU as a way of storing variable types of records in an ASCII disk file. Each item in a logical record is stored as a line of the form:

+<char>+ [<ASCII string>]<EOL>

where <char> is a single ASCII character which encodes the type of data stored on that "line", the optional <ASCII string> can be text, numbers, codes or special characters, and <EOL> is the end-of-line character(s).

2.3.4.4.1.1 ZBH Codes for Frame Header

These are coded in BaseLib.ParseFH

These codes are for the non-text information contained in the frame. Other than the protection code, this information is maintained automatically by ZOG.

ZBH Item	Representation of string in file
+A+	Frame Id String
+B+	Created by Agent Boolean
+b+	Modified by Agent Boolean
+c+	Creation Date Integer String
+M+	Modifier Name String
+m+	Modification Date Integer String
+p+	Protection Code Character
+t+	Modification Time Integer String
+U+	(List of) Frame Owners String
+V+	Frame Version Number Integer String
+Y+	(List of) Frame Accessors String (not used)
+Z+	End of Frame Header Marker

2.3.4.4.1.2 ZBH Codes for Frame Body

The code for this is in NetHandl.ParseF. The frame body consists mostly of text fields. This information is stored in Frame Title, Frame Text, Options, Local Pads Order. Within each Item of the frame, the order is Item Marker & Selection Character, Item Text, Item Position, Next Frame, Action.

ZBH Item	Representation of string in file

+ C +	Frame/Selection Comment String
+ E +	Frame/Selection Expansion Area String
+ F +	Selection's Next Frame String
+ G +	Global Pads Frame String
+ I +	Frame Text Marker String (Normally Empty)
+ L +	Local Pad Marker & Selection Character Character
+ O +	Option Marker & Selection Character Character
+ P +	Item's Position Pair of Integer Strings
+ T +	Frame/Selection Text String
+ X +	Frame/Selection Action String
+ Z +	End of Frame Body Marker
+ <other character> +	Extra Fields String

2.3.4.4.2 Procedure SetCrFH : Sets the creation information when a frame is created. Creation information includes the frames version number, protection, owners, creation date, and a field indicating if the frame was created by an agent.

2.3.4.4.3 Procedure SetModFH : Sets the modification information when the frame is modified. Modification information includes the frame version number, modifier's login name, moddate, modtime, and a field to indicate if the frame was modified by an agent.

2.3.5. Initialization routine

Initializes this module's variables, buffers and tables.

Procedure IniServ Sets the local subnet index to nil. Sets all garbage collection lists to nil. Creates temporary frame header record and buffer and a buffer for writing zeroes to a file.

2.4. ZNet

Accessing frames and subnets on a remote machine is done through routines in Module Znet. These routines are called by routines in Module ZAccessProcs whenever a frame or subnet is not located on the current machine, and must be accessed via the EtherNet. The corresponding routine in Module ZAccessProcs has nearly the same name as the Module ZNet routine, without the 'Z' preceding it.

All ZNet routines follow the same basic format. First, they set up an EtherNet request packet to send

to the target machine. Then they all call Function ZogMsg.SndRcvRecord to initiate the EtherNet communication between machines.

2.4.1. Frame Access Routines

Below are a list of the frame access routines. Three of these routines have been discussed in greater detail. For details on ZReadHeader see p. 22 For details on ZOpenFrame see p. 30 For details on ZCloseFrame see p. 34

Function ZReadFrame
Reads a frame.

Function ZReadHeader
Reads frame header information.

Function ZOpenFrame
Open a frame for modification.

Function ZCrFrame
Allows the user to create ANY specified frame .

Function ZCrNextFrame
Allows user to create the next frame in subnet.

Function ZCloseFrame
Writes and Closes a frame.

Function ZQuitFrame
Closes a frame, but will not write to frame.

Function ZEraseFrame
Deletes a frame.

Function ZUpdateFrame
Writes and closes a secondary copy on the local machine.

2.4.2. Frame Modification Routines

Function ZAddOwner
Adds a new owner to the frame

Function ZRemoveOwner
Removes an owner of a frame

Function ZSetFrProtection
Sets frame protection bits

2.4.3. Subnet Access Routines

Function ZCrSubnet
Creates a new index

Function ZClearSubnet
Clears (deletes) a subnet

Function ZGetSnInfo
Called only on the master node to get subnet information.

Function ZGetHiSubnet
Returns to the calling routine, the highest frame number in the subnet

Function ZGetNextSubnet
Generates the next subnet in the master node subnet index

2.5. ZEInt

Module ZEInt contains only four procedures, which are used for turning EtherNet interrupts Off and On.

Procedure EIntOff Turns off the EtherNet sofware interrupts.

Procedure EIntOn Turns on the EtherNet software interrupts. Calls the compiler directive InLineByte to turn off the interrupts before calling Procedure EtherInterrupt.E10Srv. Then calls Procedure E10Srv to service any interrupts which may have been sent to the current machine since the prior call to turn off the EtherNet interrupts. Thus, EtherNet interrupts are not lost when a machine has them turned off, they are merely defered.

Procedure EIntNotReady
sets a signal (local to the module) to false indicating that the machine is not yet ready to process ethernet interrupts, and turns ethernet interrupts off.

Procedure EIntReady
Sets the same signal to true, indicating the machine is ready to process ethernet interrupts, and turns the interrupts on.

2.6. ZNetServer

ZNetServer routines serve the same function on a remote machine as a Module NetHandl routine on the local machine. They are the higher level routines in frame and subnet access on the remote machine. Routines in module ZNetServer are invoked when in processing Zog, an interrupt has raised an exception on a machine. The exception is processed in the top level of Zog by the Handler Zog.E10ReceiveDone. E10ReciveDone calls Function ZNetServer. ZNetServer which acts as a dispatcher to the proper ZNetServer routine to process the request.

In general, these routines call the lower level routines to perform the request, then send back some reply to the requesting machine.

2.6.1. Frame Access Routines

These higher level routines will call routines in module ZNetProcs to perform the requested activity.

Function XZReadFrame
Reads a frame

Function XZReadHeader
Reads frame header information

Function XZOpenFrame
Open a frame for modification

Function XZCrFrame
Allows the user to create ANY specified frame

Function XZCrNextFrame
Allows user to create the next frame in subnet

Function XZCloseFrame
Writes and Closes a frame

Function XZQuitFrame
Closes a frame, but will not write to frame

Function XZEraseFrame
Deletes a frame

Function XZUpdateFrame
Writes and closes a secondary copy on the local machine

2.6.2. Frame Modification Routines

These higher level routines again, call routines in module ZNetProcs to perform the requested activity.

Function XZAddOwner
Adds a new owner to the frame

Function XZRemoveOwner
Removes an owner of a frame

Function XZSetFrProtection
Sets frame protection bits

2.6.3. Subnet Access Routines

The higher level routines which call routines in module ZNetProcs to perform the requested activity.

Function XZCrSubnet
Creates a new index

Function XZClearSubnet
Clears (deletes) a subnet

Function XZGetSnInfo
Called only on the master node to get subnet information

Function XZGetHiSubnet
Returns to the calling routine, the highest frame number in the subnet

Function XZGetNextSubnet
Generates the next subnet in the master node subnet index

2.7. ZNetProcs

All ZNetProcs routines provide an interface between higher level routines (i.e. those in NetServer) who want to access or modify a frame or subnet located on a REMOTE machine, and lower level routines which perform the actual accessing of frames and subnets. Routines here are called by routines in Module ZNetServer to carry out, whatever task, on a remote machine. Routines in ZAccessProcs carry out these tasks when frames or subnets are located on the local machine. All routines in ZNetProcs have the same names as there counterparts in Module ZAccessProcs, except that each routine name is preceded with a 'Z'. For example, the routine corresponding to ZAccessProcs.ReadFrame is ZNetProcs.ZReadFrame.

In General, these routines check the local subnet index for a subnet on the machine and perform the requested activity by calling a NetServ routine.

2.7.1. Frame Access Routines

These routines provide an interface between higher level and lower level frame access (view, create, delete) routines. Although doing very different things, they use very much the same method in locating a subnet or frame. These Functions will return success to the calling routine in Module ZNetServer if successful.

Function ZReadFrame
Reads a frame

Function ZReadHeader
Reads frame header information

Function ZOpenFrame
Open a frame for modification

Function ZCrFrame
Allows the user to create ANY specified frame

Function ZCrNextFrame
Allows user to create the next frame in subnet

Function ZCloseFrame
Writes and Closes a frame

Function ZQuitFrame
Closes a frame, but will not write to frame

Function ZEraseFrame
Deletes a frame

Function ZUpdateFrame
Writes and closes a secondary copy on the local machine

2.7.2. Frame Modification Routines

These routines use Function ZogNetServer.GetSnLocal to locate the subnet of the frame to be modified, since the subnet should already be on the machine calling a ZNetProcs routine. They then call the appropriate Module NetServ routine to perform the lower level work.

Function ZAddOwner
Adds a new owner to the frame

Function ZRemoveOwner
Removes an owner of a frame

Function ZSetFrProtection
Sets frame protection bits

2.7.3. Subnet Access Routines

Subnet Access Routines are not as similar in nature as frame access and frame modification routines. Some use routines in Module NetServ and Module ZogNetServer, while others do not. A brief summary of each of the functions follows.

Function ZCrSubnet
Updates the master node subnet index and the file :zognet>Subnet.Index

Function ZClearSubnet
Clears (deletes) a subnet

Function ZGetSnInfo
Called only on the master node to get subnet information

Function ZGetHiSubnet
Returns to the calling routine, the highest frame number in the subnet

Function ZGetNextSubnet
Generates the next subnet in the master node subnet index

2.8. ZOGMsg

Routines in module ZogMsg are called when communication is necessary between machines. These routines send and receive ethernet request and reply packets. Each of the send and receive routines has its own local handler, E10ReceiveDone, which is invoked when an interrupt is generated on a machine due to the synchronous communication between machines. E10ReceiveDone examines the Ethernet Handler State, and by it, controls execution of the routine.

- Send Routines

- Receive Routines
- Message verification and handling routines
- Utilities
- EtherNet Handler States

2.8.1. Send Routines

These routines send replies or acknowledgements to another machine.

2.8.1.1 Function SndRcvRecord

For details on the inner workings of ZOGMsg.SndRcvRecord see p. 23.

2.8.1.2 Function ZogMsg.SendRecord

Sends a record to a remote machine and waits for an acknowledgement of receipt of the record.

2.8.1.2.1 Sets addresses to be correct, in various records, so that the record can be received on the remote machine.

2.8.1.2.2 Resend Loop

At this point the SendRecord enters a loop to send a request to the other machine, saying, "Well, Go ahead". The loop will attempt to send the request a maximum of NumberResends times (5). To send the request, first the ethernet interrupts are turned off. Next, a call is made to Procedure Ether10IO.E10WIO which starts an EtherNet I/O operation and waits for it to complete. In this case, information is being sent, so E10WIO makes sure the information is sent over the EtherNet.

2.8.1.2.3 If an error is detected in sending the message, then exit SendRecord. Otherwise, set the EtherNet Handler State to indicate that the local machine is waiting for the acknowledgement from the remote machine (SWaitAck) and turn on the EtherNet interrupts.

2.8.1.2.4 Got Acknowledgement Time-Controlled Loop

If the acknowledgement is received by the machine sending the message, an interrupt is generated, causing an exception to be raised by the EtherNet MicroCode, thus invoking the local Handler E10ReceiveDone. E10ReceiveDone sees that the EtherNet Handler State indicates that the local machine is waiting for an acknowledgement (SwaitAck), and signals acknowledgement by assigning the EtherNet Handler State to be that of 'Got the Acknowledgement' (SGotAck). If the acknowledgement is received, exit SendRecord.

2.8.1.2.5 If after five attempts no acknowledgement is received from the remote machine, then exit SendRecord with an error.

2.8.1.3 Function SendBuffer

For detials on the inner workings of ZOGMsg.SendBuffer see p. 36.

2.8.2. Receive Routines

Recieve routines receive requests and acknowledgements.

Function ReceiveRecord (For Details see p. 38).

Function ReceiveBuffer (For Details see p. 28).

2.8.3. Message verification and handling routines

For details on ZOGMsg.HandleMsg see p. 23.

Function ChkMsg Checks a message received to see if it is a valid ethernet message.

Function HandleMsg
Boolean function that returns true if there is a valid ZOG request.

Function AnotherMsg
Tests if another message has been received and is waiting to be used inside the handler, before exiting the handler. It is necessary to perform this check and process any additional messages before exiting the handler, because the interrupt for the additional messages has already occured, and was handled by an empty nested handler. If this is not done the message would be ignored until another new message was received.

2.8.4. ZOGMsg Utilities

Function SwapByte
Used to swap two bytes of a word

Function CvIntStr Converts an integer to a decimal string

Function EqAddr Used to test if two ether net addresses are equal; this is used to make sure a reply or acknowledgement comes from the machine that it was supposed to

Procedure SuspendZCGMsg
Resets the network and removes any pending receives

Procedure ResumeZOGMsg
resets the network and reposts all receives

Function InitZOGMsg
Initializes the ethernet and allocates all Ethernet buffers

Procedure ResetZOGMsg
Resets the ethernet and deallocates all buffers

Functions GetMyAddr and GetMyName
Get address and name of this machine

Procedure RePostReceive
Reposts a receive with buffers of the msg just received

2.8.5. EtherNet Handler States

Ethernet Handler States refer to the state of a particular machine which is involved in ethernet communication with another machine. The 'state' of a machine can be, for example, that it is waiting for an acknowledgement or that the machine has gotten a reply. It describes the status of the communication between machines.

2.8.5.1 Common State : SNotReady (always the initial state)

2.8.5.2 ProbeName States : SWaitProbe, SGotProbe (for reply to Probe msg)

2.8.5.3 SendRecord States : SWaitAck, SGotAck (for acknowledge of sent record)

2.8.5.4 ReceiveRecord States : SWaitRcv, SGotRcv (for msg to be received)

2.8.5.5 SndRcvRecord States : SWaitSndAck, SWaitReply, SGotReply (Got Ack. and waiting for reply, got reply)

2.8.5.6 SendBuffer States.

SWaitGo	Waiting for the go ahead message.
SSendFirst	Got Go ahead, sending the first buffer.
SWaitBufAck	Waiting for an acknowledgement of a sent buffer.
SSentAll	All buffers have been sent and acknowledged.

2.8.5.7 ReceiveBuffer States : SWaitGoAck, SReceiving, SGotAll (Waiting for ack to go ahead, receiving buffer, got all information

2.9. ZOGMsgDefs

Module ZOGMsgDefs defines the constants and record types used in module ZogMsg as ethernet request and reply packets(records). The naming convention of the request packets is to end the pointer type with '0PTyp', while the reply packets end in '1PTyp'. In the case of the close frame reply packets there is a second packet ending in '2PTyp'. Examples of ethernet request and reply packets are below.

2.9.1. Ethernet Request packet records

Module ZogMsgDefs contains the declarations of all of the Ethernet request and reply packets. For the most part the types contained in this module are the same with small variations in the records, due to the function of a particular type. Only two of the types will be shown here. The information contained in these records allows the sending and receiving routines to accomplish their purpose (i.e. opening, closing, etc.). This information is loaded into the appropriate record, recast to a message or buffer type and sent over the ethernet. On the remote machine, the appropriate receiving routine knows the structure of the information in the request or reply packet, so it knows where to get the information it needs.

2.9.1.1 Open Frame Request Packet.

OpnF0PTyp = ↑OpnF0Typ

OpnF0Typ = packed record

Id	integer; Constant identifier in ZogMsgDefs
LocalAddr	EthernetAddress;
RemoteAddr	EtherNetAddress;
RemoteName	String15;
GR	GeneralReturn; used in return packet
Name	UsrIdTyp;
AgentFlag	Boolean;
Sid	SidTyp; Subnet ID
PrimeNode	NodeTyp;
SecCnt	integer;
SecNodes	SNodesTyp;
FrNum	integer;

2.9.1.2 Open Frame Reply Packet

OpnF1PTyp = ↑OpnF1Typ

OpnF1Typ = packed record

Id	integer; Constant identfier from ZogMsgDefs
LocalAddr	EthernetAddress;
RemoteAddr	EthernetAddress;
RemoteName	String15;
GR	GeneralReturn; Return code from remote node

FHBCnt long; Count of Frame Header pages

FBCnt long; Count of Frame Body pages

2.9.1.3 Close Frame Request Packet

ClsF0PTyp = ↑ClsF0Typ

ClsF0Typ = packed record

Id integer; Constant identifier in ZogMsgDefs

LocalAddr EthernetAddress;

RemoteAddr EtherNetAddress;

RemoteName String15;

GR GeneralReturn; used in return packet

Name UsrIdTyp;

AgentFlag Boolean;

Sid SidTyp; Subnet ID

FrNum integer;

FBCnt long; Count of Frame Body Pages

2.9.1.4 Close Frame Reply Packet 1

ClsF1PTyp = ↑ClsF1Typ

ClsF1Typ = packed record

Id integer; Constant identfier from ZogMsgDefs

LocalAddr EtherAddress;

RemoteAddr EtherAddress;

RemoteName String15;

GR GeneralReturn; Return code from remote node

2.9.1.5 Close Frame Reply Packet 2

ClsF2PTyp = ↑ClsF2Typ

ClsF2Typ = packed record

Id integer; Constant identfier from ZogMsgDefs

LocalAddr EthernetAddress;

RemoteAddr EthernetAddress;

RemoteName String15;

GR GeneralReturn; Return code from remote node

FHBCnt long; Count of Frame Header pages

2.10. ZOGNetServer

Routines in Module ZogNetServer deal with subnet access on local and remote machines and the building and maintenance of the local subnet database. These routines are called from routines in module ZAccessProcs whenever frames of subnets or subnets themselves are being accessed. It also builds and maintains the network servers database indicating the status of the other machines in the ZOG network.

2.10.1. Subnet Locating Routines

Function GetSnLocal

Takes a subnet name and hashes into the local subnet database and find the correct subnet. Returns a boolean value of true if found. A pointer to the subnet record is returned in a variable parameter.

Function GetSnMaster

Tries to get information on a subnet by requesting it from the master node. This will be called when the subnet is not found in the local subnet database and a machine needs to know on what machine the subnet can be found. The function returns a GeneralReturn type, indicating success or failure. Via parameters, it also returns the primary node, and the number and identity of the secondary nodes.

Function GetSnRecord

Hashes into the local subnet database to find a subnet. If not found there, it looks in the subnet database of the master node. A side effect of a call to GetSnRecord is to read the file if it is located on the local disk. Returns a GeneralReturn type indcating success or failure.

Function ChkSnRecord

Is called by routines in ZNetProcs when a machine is being accessed for a subnet, therefore the information is assumed to be on the local machine. It returns the same information as GetSnMaster and will attempt to read the file if it is located on the local disk.

2.10.2. Subnet Maintenance Routines

Function CrSnRecord

Creates a subnet record and enters it into the subnet hash table (subnet database). Returns a GeneralReturn type and a pointer to the new entry.

Procedure ErSnRecord

Deletes a subnet record from the subnet hash table in all but the master node index. This is used to force the next call of Function GetSnRecord to go to the master node for the information.

Function AddSnRecord

Inserts a new entry into the local subnet database.

Function OpnSnRecord

Takes a pointer to a subnet record from the local subnet database and opens the local subnet file through a call to Function NetServ.OpnSn--. The act of opening is merely setting a boolean field of the subnet record type.

Function UpdateIndex
Is used to update the Subnet.Index file which stores the list of all subnets.

Function EnterSubnet
Is used to enter a new subnet into the subnet database on the master node and to update the Subnet.Index file

Procedure BuildSubnets
Is called by Procedure InitZogNetServer to construct the local subnet index.

2.10.3. Server Routines

Procedure BuildServers
Is called by Procedure InitZogNetServers during the initialization of Zog to build the servers index consisting of the machines on the network.

Function Probe Is used to test another network node, to see if it is actually up and running in Zog. If that node is found to be up, via a call to Function ZogMsg.ProbeName, the servers index is updated to indicate the node is up in Zog.

2.10.4. ZogNetServer Utility Routines

Procedure CvStrUpper
Converts a string to all upper case

Procedures MsgError and PrintError
Used to output error messages

Function SnHash Is the hashing function for the subnet database

Function NxtSn Generates the next subnet name in the master nodes subnet database. If it receives an empty string it returns the first entry in that table and if it returns the empty string, then there were no more entries in the masters subnet index.

Procedure InitZogNetServer
Initializes the data structure for the ethernet and the ethernet itself so that it is prepared to receive requests from other machines. It also call the routines to build the subnet and servers indexes.

3. ZOG Editor Modules

3.1. ZED Modules

ZCrFrame	Procedures to Create Frames via tDi action or TDFC
ZDspInc	Low-Level Display Utilities and globals for ZED and SLED
ZEdDefs	ZED (and SLED) TypeDefs and Global Variables
ZEdFram	Full-Frame level ZED Commands (Upper Case Commands)
ZEdIt	Main ZED Module - Command Parsers, hidden area commands
ZEdItem	Per Item ZED Commands (Lower Case Commands)
ZEdNew	Module to implement mouse selection of text within frames (Not Currently Used)

ZEdUtil	Low-Level utility routines for ZED

3.2. SLED Modules

ZBrEd	Special-purpose extension of SLED for AirPlan Frame Editing - Provides routines to break apart/put together AirPlan slots on specialized AirPlan input frames
ZEnvEd	Main Environmental Editor Module - Contains higher level slot editing procedures
ZEnvUtil	Contains lower level slot editing utilities
ZSled	Main SLED Module - SLED Command Parser, AirPlan utilities
ZSledUtil	Contains lower level SLED support functions

4. ZOG Agents Modules

4.1. Planning and Evaluation (Task Management) Agents

AgAdjDt	Adjusts the dates and times in a specific task tree
AgGenr	Creates a generic task tree from a specific task tree
AgGreen	Submit task to Green Sheet
AgInst	Instantiates a specific task tree from a generic task tree
AgInTask	Initializes a specific task tree
AgTPlan	Creates a task plan from a specific task tree in disk file form for outputting to a hard copy device
AgUpTask	Updates a task tree "upward" to propagate leaf node changes
AgZPlan	Creates a task plan from a specific task tree in a new tree of ZOG frames
AgPlan	Creates a task plan from a specific task tree in disk file form for outputting to a hard copy device

4.2. Backup and Transport Agents

These agents are used by system maintainers for reformatting subnets for backup and transportation

AgArchive	Archive a subnet or frame to a floppy
AgBackup	Write zbh for all perqs
AgBak	Write zbh for all subnets modified since a specific date and time for a specified Perq
AgVBH	Write Perq ZOG frames in VAX zbh format
AgZBH	Write zbh format of Perq ZOG frames

4.3. ZOG Special function Agents

4.3.1. Writing frames in a form suitable for printing

AgDoc Write a tree of frames into a disk file using a format suitable for printing

AgPic Write a single frame into a disk file without changing the format

4.3.2. Saving old versions of frames

AgOld Copies a frame, linking the copy to the frame through an Old local pad

AgPost Saves the current version of a frame as an Old frame, then clear the frame ,next copy the schema of the 0th frame to the current frame

4.3.3. Utilities

AgHiSubNum Vinson utility routines

AgLink Links an option to the frame in the other windo in an accessor-like manner. (Experimental)

AgMessage Send a message to another Perq

AgTest Schema for creating new agents

AgCode Create a text file ready for compiling from a code subnet

4.3.4. Fonts and Graphics

AgBar Creates a bar graph from a given data frame

AgRFont Changes the fonts for a given subnet

4.3.5. Creating an index or directory of subnets

AgAlphaSNL Creates an index of subnets either alphabetically or by Perq

AgDir Creates a directory for subnets on a Perq or all Perqs

AgIndex Creates an alphabetical index to subnets on a Perq or all Perqs

4.4. Subnet Repair and Updating Agents

AgMerge Standardizes a subnets local pads to that of a given schema frame

AgPar Corrects all bad parent and top links

AgProt Modifies the protection on frames

AgSwap Global string replacement

AgOwn Adds or deletes the owner of a frame or frames

AgChkSecond Checks secondary copy of a subne

4.5. SORM and Weapons Elevator Agents

The SORM and Weapons Elevator Agents are very specialized. Most of the following agents are used in formatting the document that is produced when the SORM and Weapons Elevator subnets are written out.

4.5.1. AgDgm : Writes out a chapter of diagrams

This agent will print, in scribe compatable format, a tree of frames. The format is for diagrams and GAPL(Government Allowance Parts List). Each frame corresponds to one picture and each picture may have a GAPL associated with it.

4.5.2. AgGAPL : Prints a tree of frames in scribe compatable format

This agent will print in scribe compatable format, a tree of frames. The format is for GAPL(Government Allowance Parts List) and prints a depth first search list of all parts in the tree.

4.5.3. AgMgmt : Produces a listing of all the frames title text

This agent will produce a depth first listing of all the frames title text in addition to a cross reference to the current frame. It is currently being used to generate the Apendix for the management codes in the *organization* section of the ships SORM.

4.5.4. AgOpr : Prints a tree of fraems in depth first search order

This agent will print a tree of frames in depth first search order. It is intended to print the *operate* section of the Weapons Elevator Manual. It's main features are that it prints out title text and a mini table of contents for each frame that has options.

4.5.5. AgOrg : Prints lists of responsibilities of each billet

This agent will print a tree of frames in depth first search order. It is intended to print the *organization* section of the ship's SORM. It prints a list of responsibilities of each billet with a cross reference into the task net where it is defined.

4.5.6. AgTask : Prints out option text for each frame that has options

This agent will print a tree of frames in depth first search order. It is intended to print the *understand* section of the Weapons Elevator Manual and the *operate and maintain* section of the ship's SORM. It's main features are that it prints out option text for each frame that has options. It will also print a mini table of contents if the frame has the keyword "CONTENTS" in the frame comment area.

4.5.7. AgText : Prints out the frame text on each frame visited

This agent will print a tree of frames in depth first search order. It simply prints the frame text on each frame visited. In addition it will follow any ">.More" local pads and follow any option tree that exits on the "More" frame.

4.5.8. AgThy : Prints out theory section of Weapons Elevator Manual

This agent wil print a tree of frames in depth first search order. It is intended to print out the *theory* section of the Weapons Elevator Manual and the ship's SORM. Option text is printed as the first sentence of a paragraph with any frame text on the next frame appended to the end. Each succeeding level is treated as a subparagraph of the proceeding paragraph and is indented as in an outline. No local pad cross references are generated in this agent.

4.5.9. AgTrb : Prints out troubleshooting section of Weapons Elevator Manual

This agent will print out a tree of frames in depth first search order. It is intended to print the *troubleshooting* section of the Weapons Elevator Manual. It's main features are that it prints out title text and a mini table of contents for each frame that has options. It also generates "subchapter", "section", "subsection", and "paragraph" commands for the first 4 levels in the tree. Each successive level has the title text printed in bold face type.

4.5.10. AuxOrg : Prints out the appendixs for the ship's SORM

This agent will print out a tree of frames in depth first search order. It is intended to print the appendixs in the ship's SORM for parts of the organization such as department heads, division officers, leading chiefs, etc. Its only difference from AgOrg is that it will first mark a tree of frames as having already been seen so that duplication will be avoided when only a partial list is desired. As a side effect of having seen a frame before a cross reference is generated. In this way the list in an appendix of division officers will point to the correct location in the organization chapter. This is really somewhat of a kludge since we cannot keep the frames vid from running AgOrg around for a seco visited from running AgOrg around for a second run.

4.6. Agents Libraries

- AgentLib
- ArchLib
- EnvLib
- FormLib
- FramLib

- FSelLib
- PlanLib
- SelLib
- StackLib
- ZFileIO

4.7. Shell Utility Modules

- ZCopy
- ZCSDXNet
- ZDelete
- ZDirect
- ZMount
- ZPath
- ZRemotePrint
- ZRename
- ZSearch
- ZStdError
- ZTypeFile

4.8. Agent/Shell Utility Invocation Modules

- ZAgent
- ZXAAgent
- ZXBAgent
- ZShell
- ZXShell

5. ZOG AirPlan Modules

These modules are being maintained on board the USS CARL VINSON.

- AirCom
- AirDefs
- AirLib
- AirOutput
- ApChkMess
- ApFlOver
- ApLOver
- ApOpsFile
- ApPlOver
- ApPagePac
- ApROver
- ApSetEvt
- ApVerifyOut
- ZXAirPlan

6. PERQ Operating System Modules Imported by ZOG

Many modules from the PERQ Operating System are used throughout ZOG. These modules export procedures for string manipulation, for memory allocation, raster ops, ethernet interrupt handling and the like.

CmdParse	Provides a number of routines to help with command parsing.
Dynamic	Implimements Pascal dynamic memory allocation (New and Dispose)
Ether10IO	Provides the interface to the 10 Mbaud Ethernet microcode.
EtherInterrupt	Provides the interrupt service for the Ethernet.
Except	Provides the exception routines.
FileSystem	Provides the File System routines.
IO – Others	Provides routines for the Cursor, Table, Screen, Time a& Keyboard

IO – Private	Exports interrupt routines and definitions which are private to the modules which make up the IO system.
IO – Unit	Provides procedures to perform IO on the various IO devices.
Memory	Memory is the PERQ memory manager
PERQString	Impliments the string manipulation routines for PERQ Pascal.
Screen	Provides the interface to the PERQ screen including multiple windows
Stream	Impliments low-level Pascal IO. It is called by higher level routines such as Reset, ReWrite, Get, Put.
System	Initializes POS and goes into a loop alternately running Shell and ZOG.

INITIAL DISTRIBUTION

Copies	
1	USS CARL VINSON
1	ONR/270
12	DTIC

CENTER DISTRIBUTION

Copies	Code	Name
1	18	G. Gleissner
1	1808	D. Wildy
1	182	A. Camara
1	1826	J. Garner
1	1826	J. Jeffers
10	1826	D. Schmelter
1	522.1	TIC (C)
1	522.2	TIC (A)
1	93	L. Marsh

DTNSRDC ISSUES THREE TYPES OF REPORTS

1. DTNSRDC REPORTS, A FORMAL SERIES, CONTAIN INFORMATION OF PERMANENT TECHNICAL VALUE. THEY CARRY A CONSECUTIVE NUMERICAL IDENTIFICATION REGARDLESS OF THEIR CLASSIFICATION OR THE ORIGINATING DEPARTMENT.

2. DEPARTMENTAL REPORTS, A SEMIFORMAL SERIES, CONTAIN INFORMATION OF A PRELIMINARY, TEMPORARY, OR PROPRIETARY NATURE OR OF LIMITED INTEREST OR SIGNIFICANCE. THEY CARRY A DEPARTMENTAL ALPHANUMERICAL IDENTIFICATION.

3. TECHNICAL MEMORANDA, AN INFORMAL SERIES, CONTAIN TECHNICAL DOCUMENTATION OF LIMITED USE AND INTEREST. THEY ARE PRIMARILY WORKING PAPERS INTENDED FOR INTERNAL USE. THEY CARRY AN IDENTIFYING NUMBER WHICH INDICATES THEIR TYPE AND THE NUMERICAL CODE OF THE ORIGINATING DEPARTMENT. ANY DISTRIBUTION OUTSIDE DTNSRDC MUST BE APPROVED BY THE HEAD OF THE ORIGINATING DEPARTMENT ON A CASE-BY-CASE BASIS.